2. 딥러닝을 위한 수학

- MSE(Mean Squared Error)를 구현으로 이해한다.
- SGD (Stochastic Gradient Descent)을 구현으로 이해한다.
- 역전파를 구현으로 이해한다.
- 선형회귀를 구현으로 이해한다.
- 시그모이드 함수를 구현으로 이해한다.
- 로지스틱 회귀를 구현으로 이해한다.

2. 딥러닝을 위한 수학

<u>2.1 MSE(Mean Squared Error)</u>

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

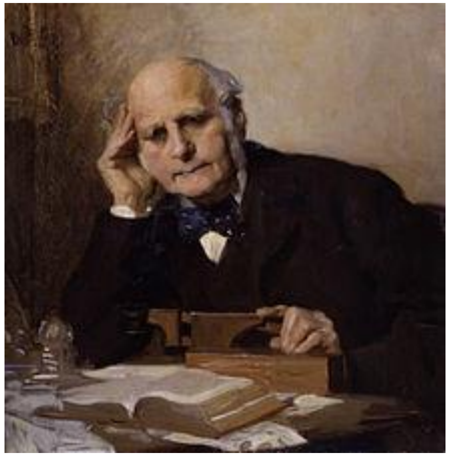
2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

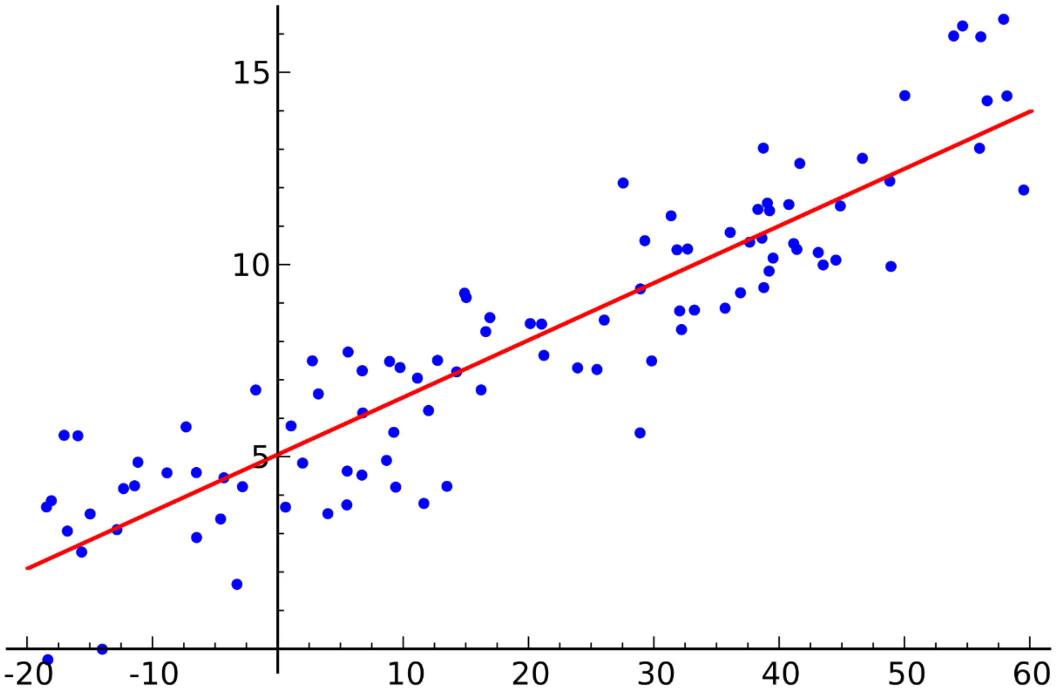
Regression - 회귀

"Regression toward the mean"



Sir Francis Galton
(1822 ~ 1911)

Linear Regression – 선형 회귀

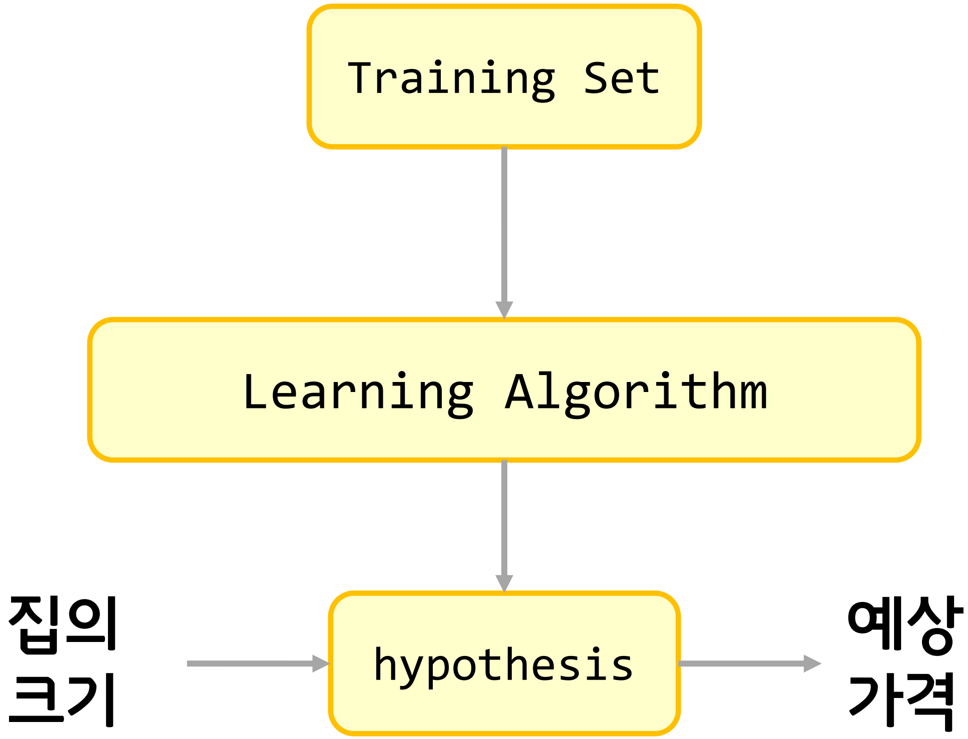


https://en.wikipedia.org/wiki/Linear_regression

$$y = ax + b$$

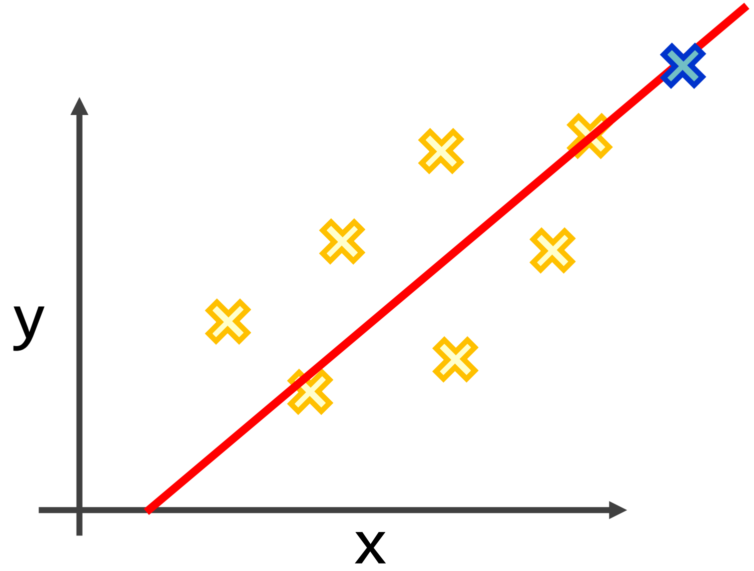
$$y = wx + b$$

Cost function



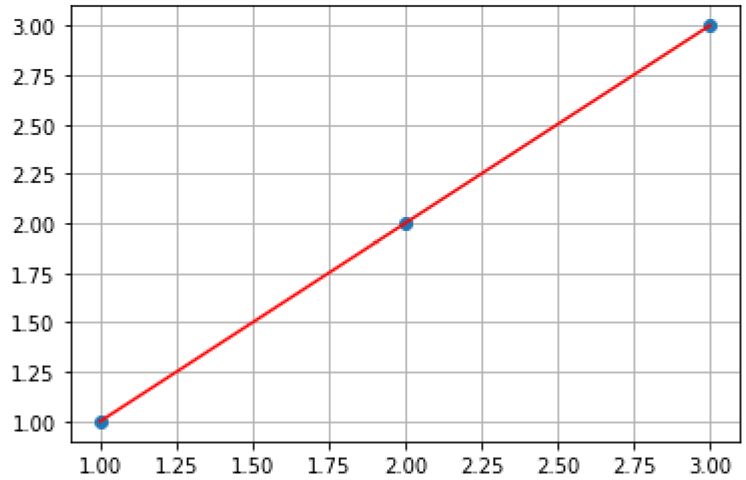
hypothesis ? 가설

$$\hat{y} = wx + b$$

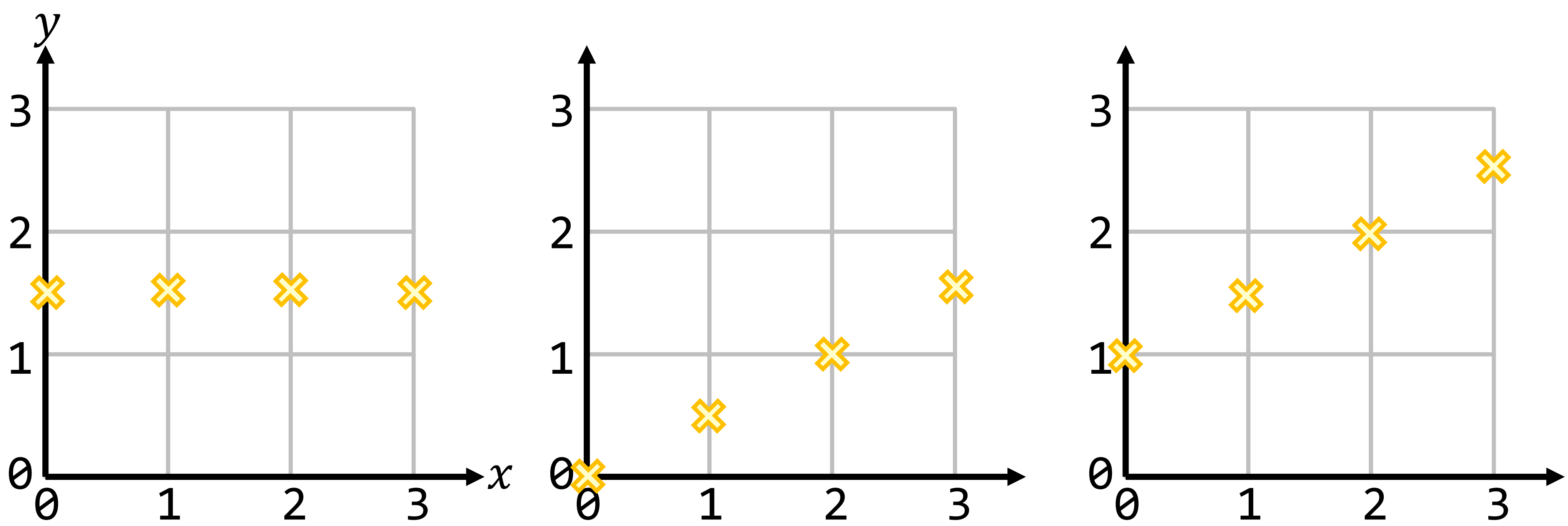


```
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
import numpy as np
import matplotlib.pyplot as plt

X=np.array([1,2,3,4,5])
Y=np.array([1,2,3,4,5])

model=Sequential()
model.add(Dense(1, input_dim=1, activation='linear'))
sgd=optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd ,loss='mse',metrics=['accuracy'])
model.fit(X,Y, batch_size=1, epochs=10, shuffle=False)

plt.plot(X, Y, 'bo')
plt.plot(X, model.predict(X), 'r-')
plt.show()
```

```
import numpy as np   # numeriacal computing
import matplotlib.pyplot as plt  # plotting core

x = np.array([1,2,3])
y = np.array([1,2,3])

plt.plot(x,y, 'o' )
plt.plot(x,y, 'r-' )

plt.grid(True)
plt.show()
```



$$\hat{y} = wx + b$$



w = 0
b = 1.5

w = 0.5
b = 0

w = 0.5
b = 1

Hypothesis :

$$\hat{y} = wx + b$$

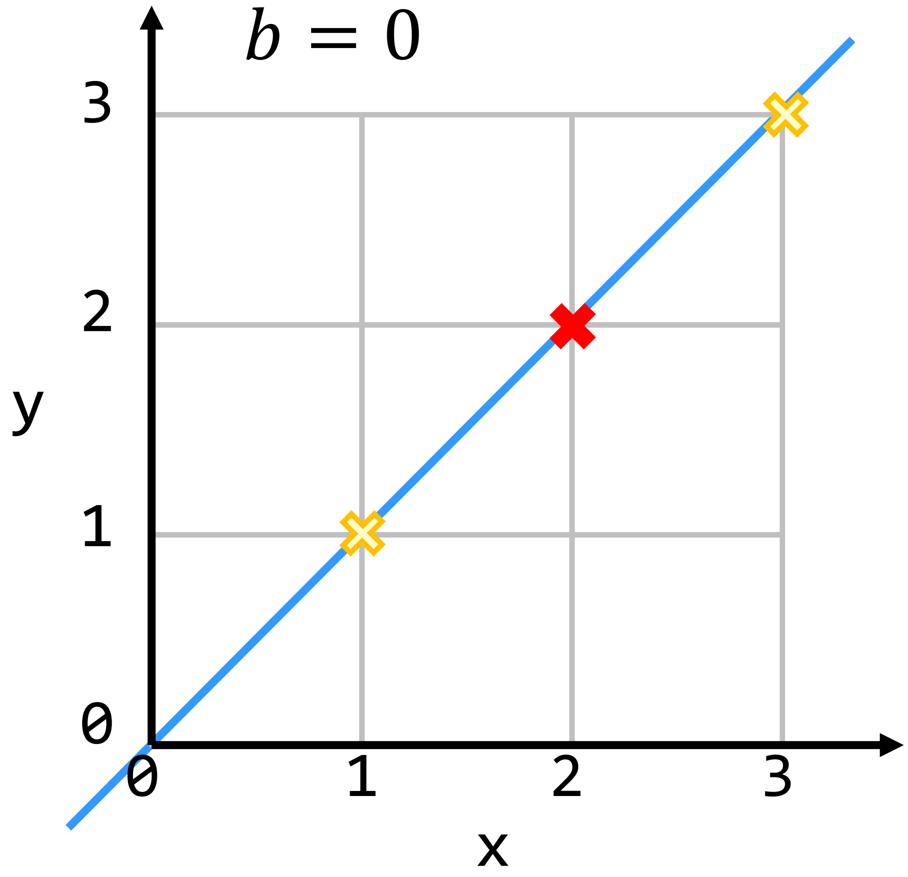
Parameters:

$$w, b$$

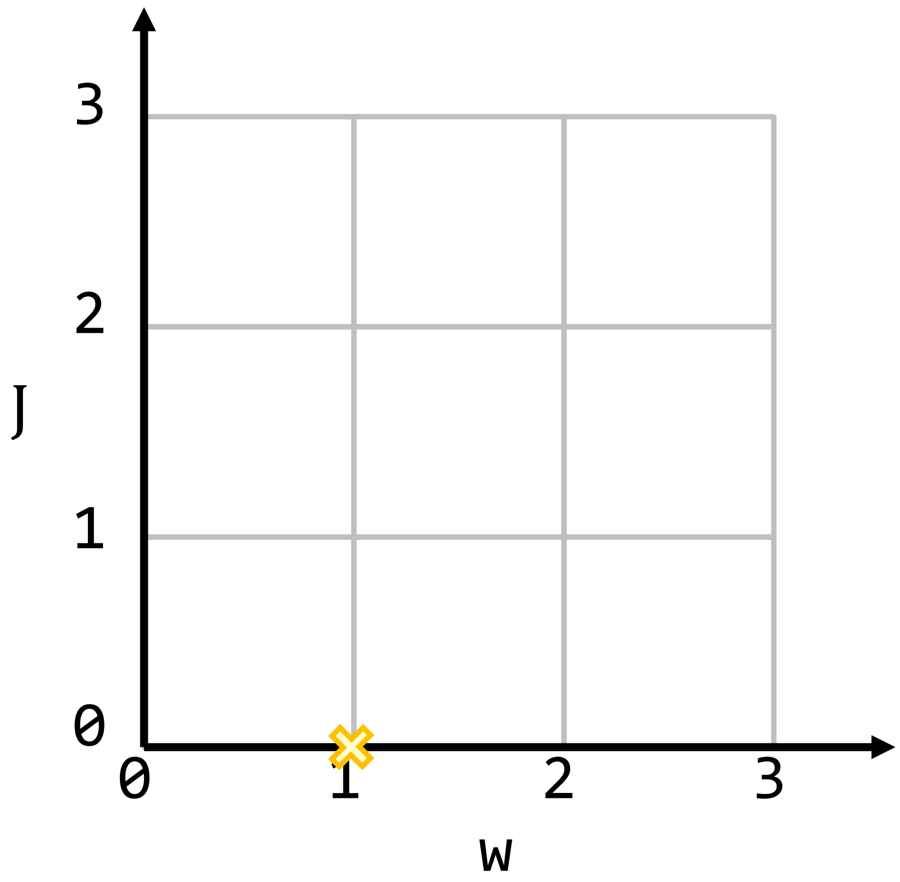
Cost Function

$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 0$$

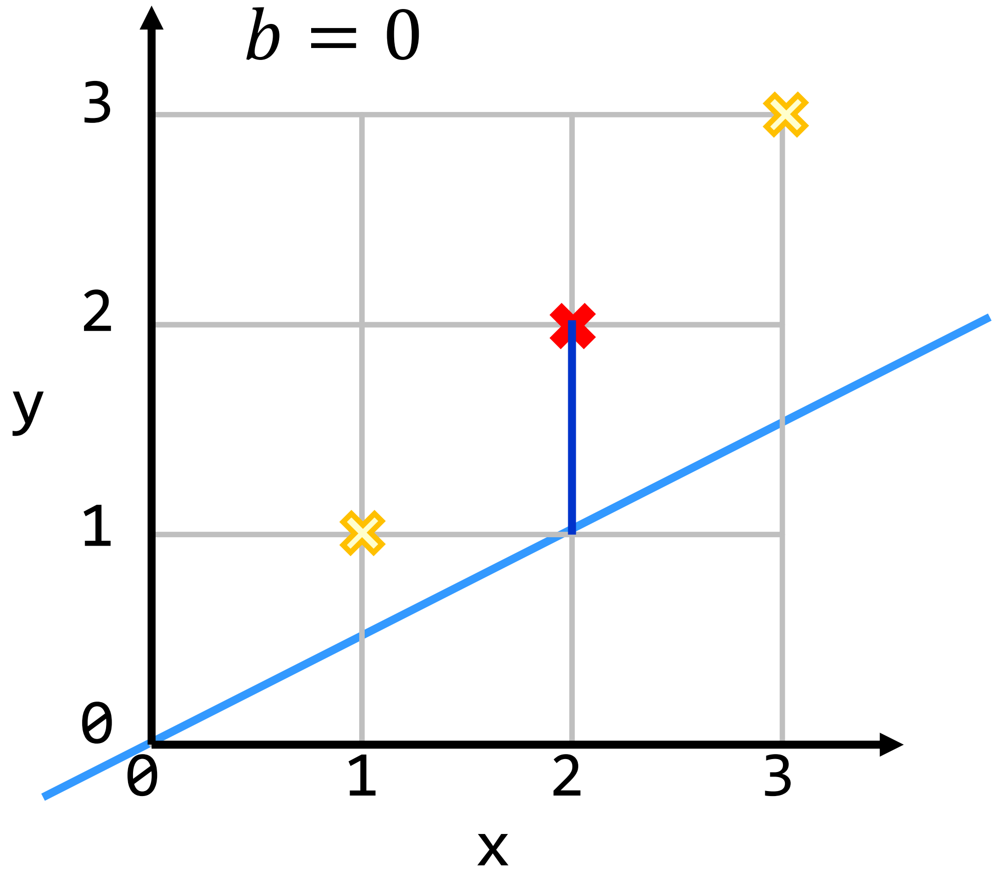


$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

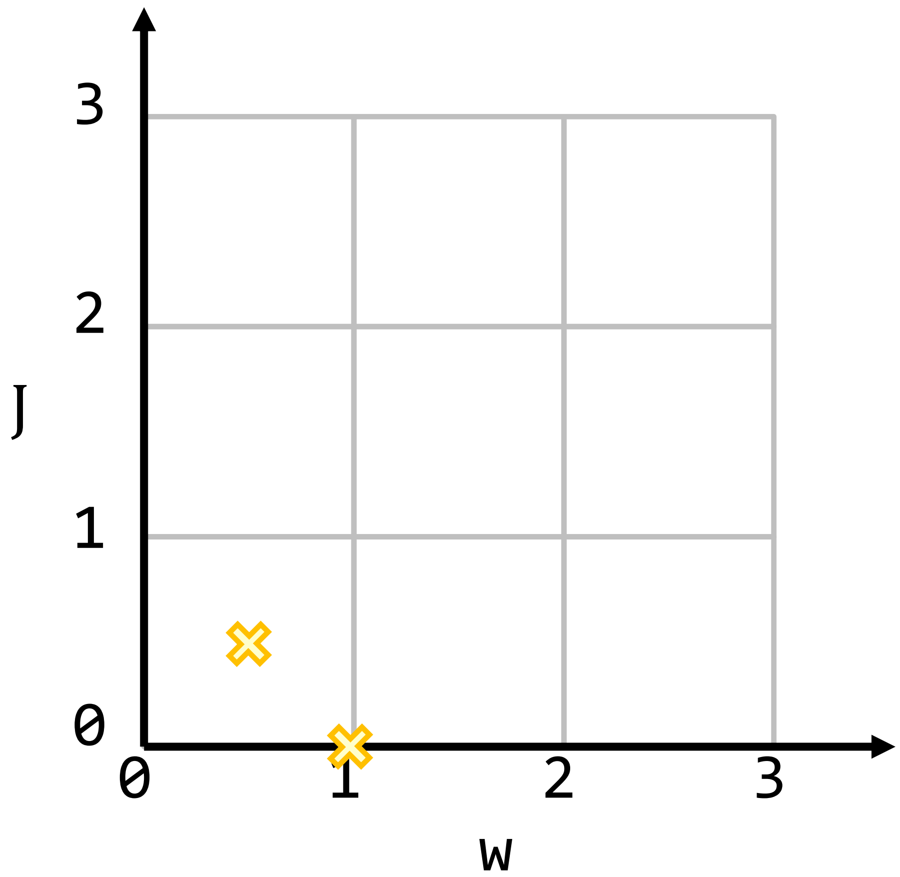


Cost function

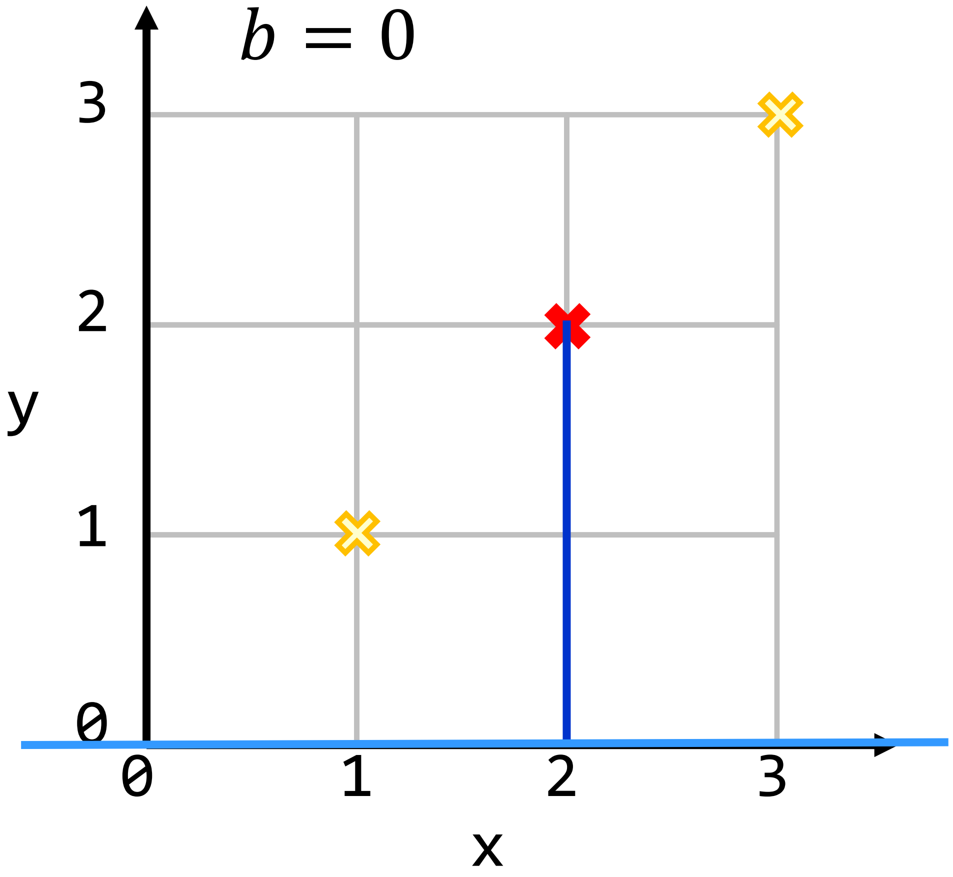
$$\hat{y} = wx + b$$
$$w = 0.5$$
$$b = 0$$



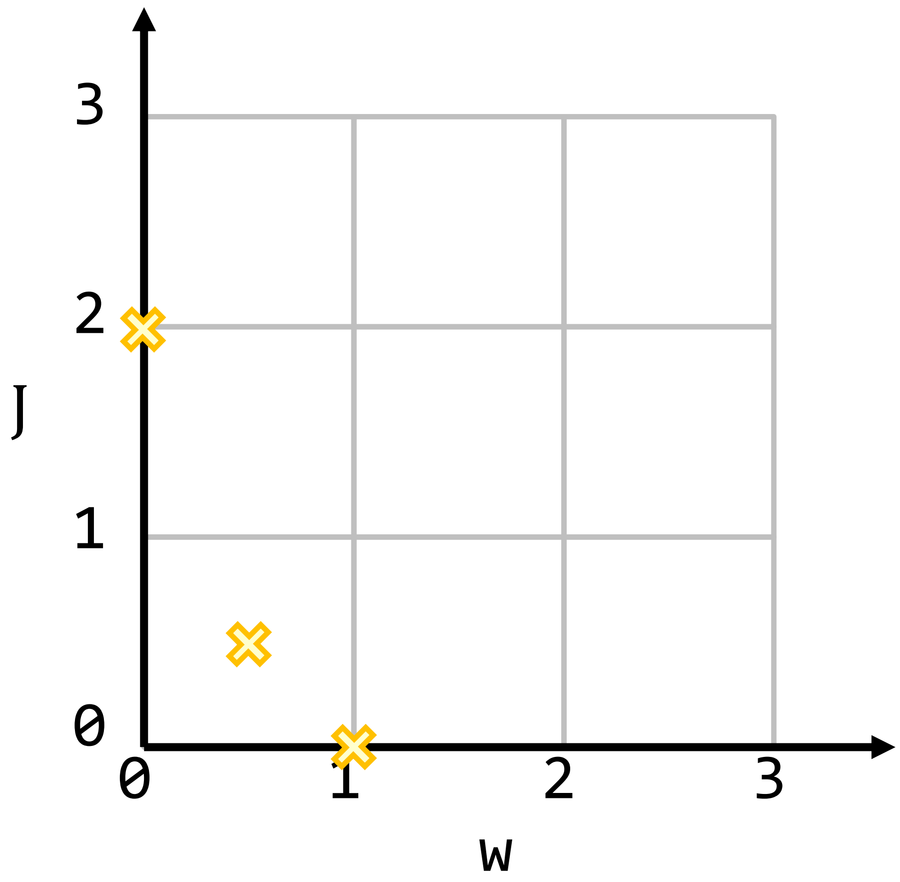
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



$$\hat{y} = wx + b$$
$$w = 0$$
$$b = 0$$

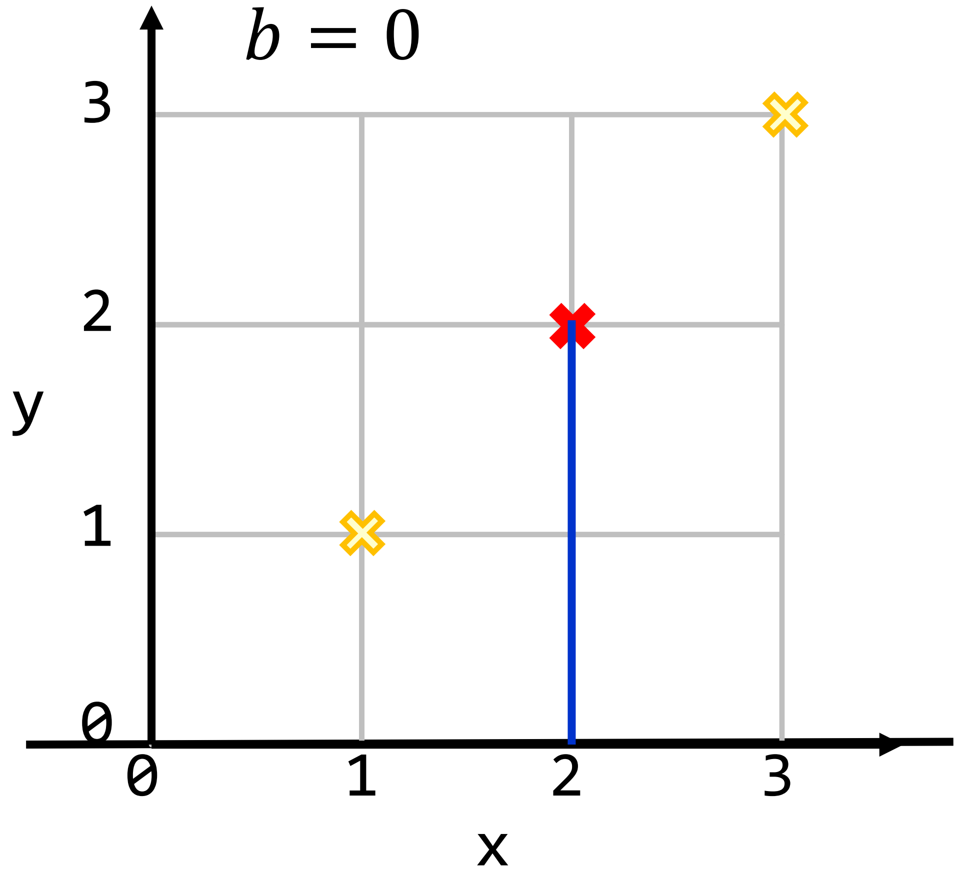


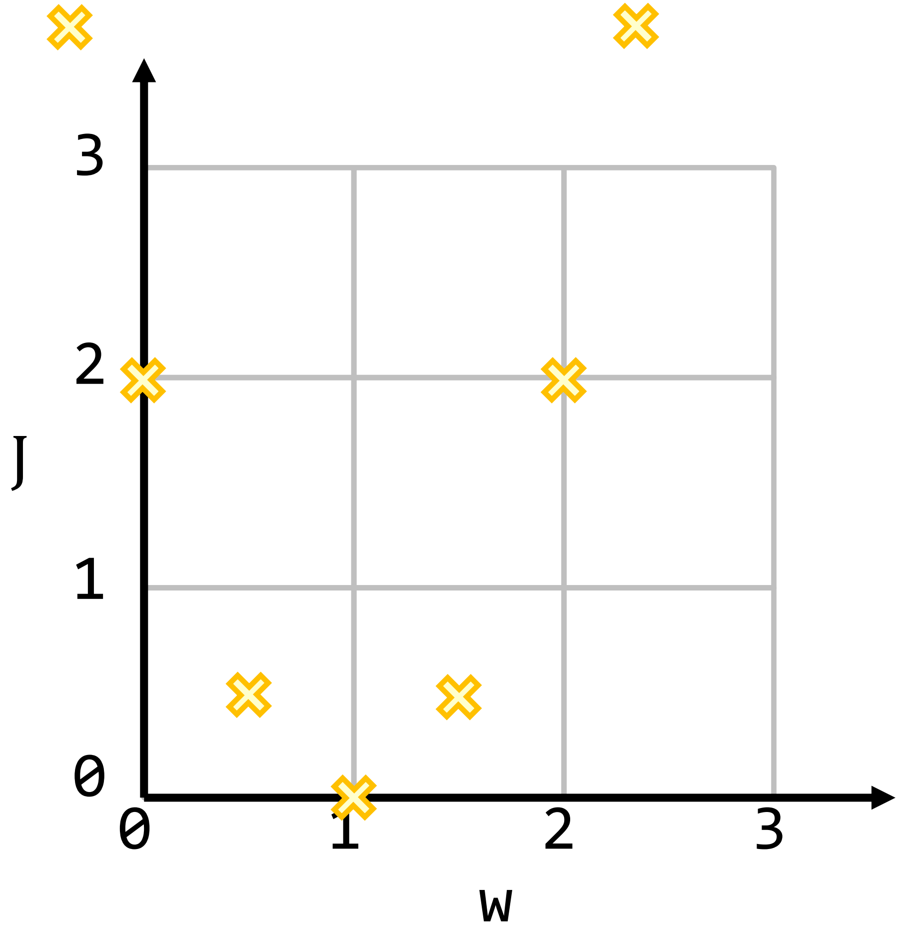
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



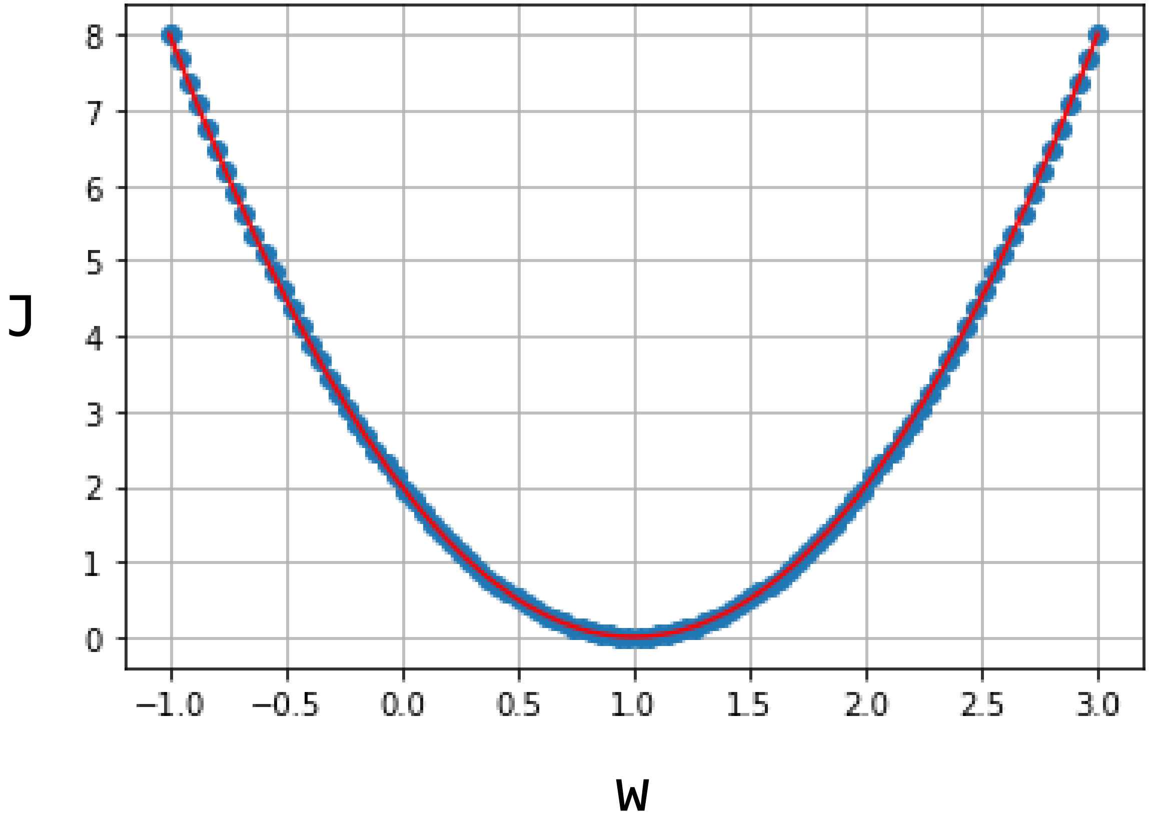
$$\hat{y} = wx + b$$
$$w = 0$$
$$b = 0$$

$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

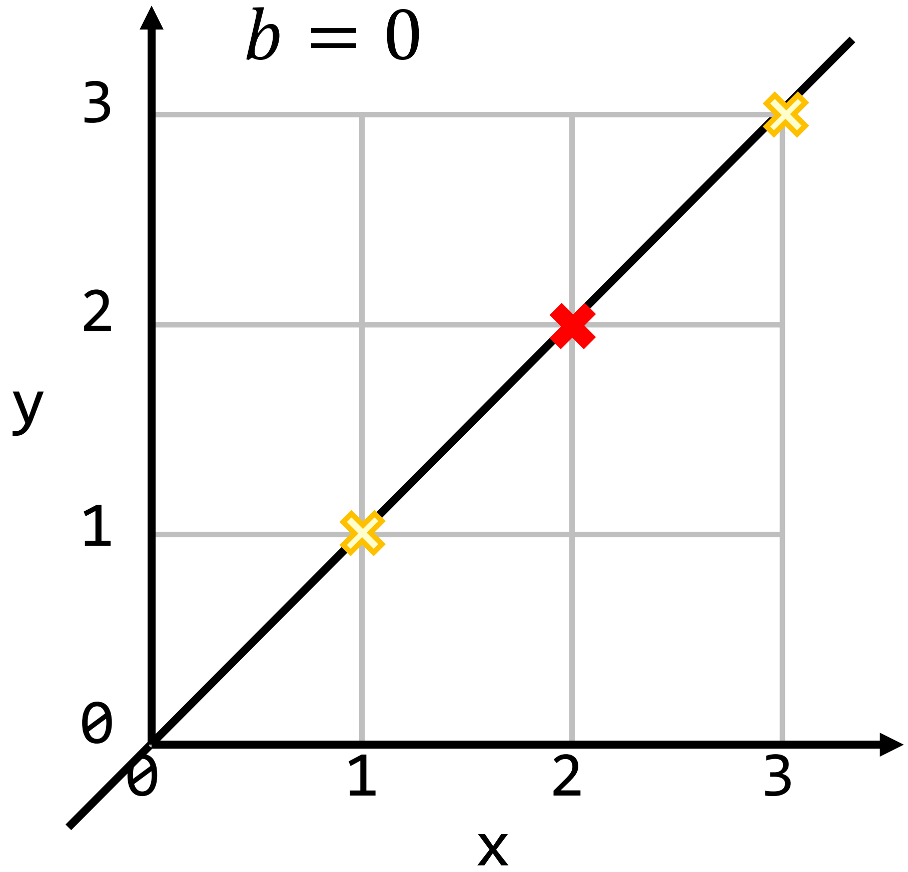




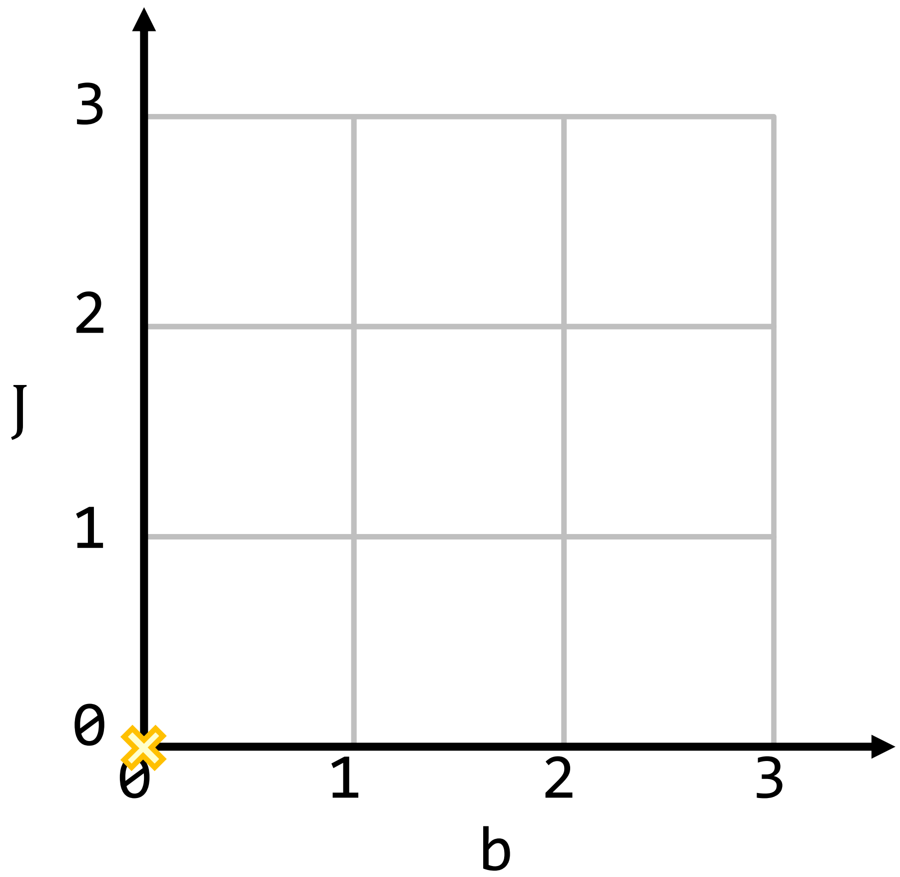
Cost function



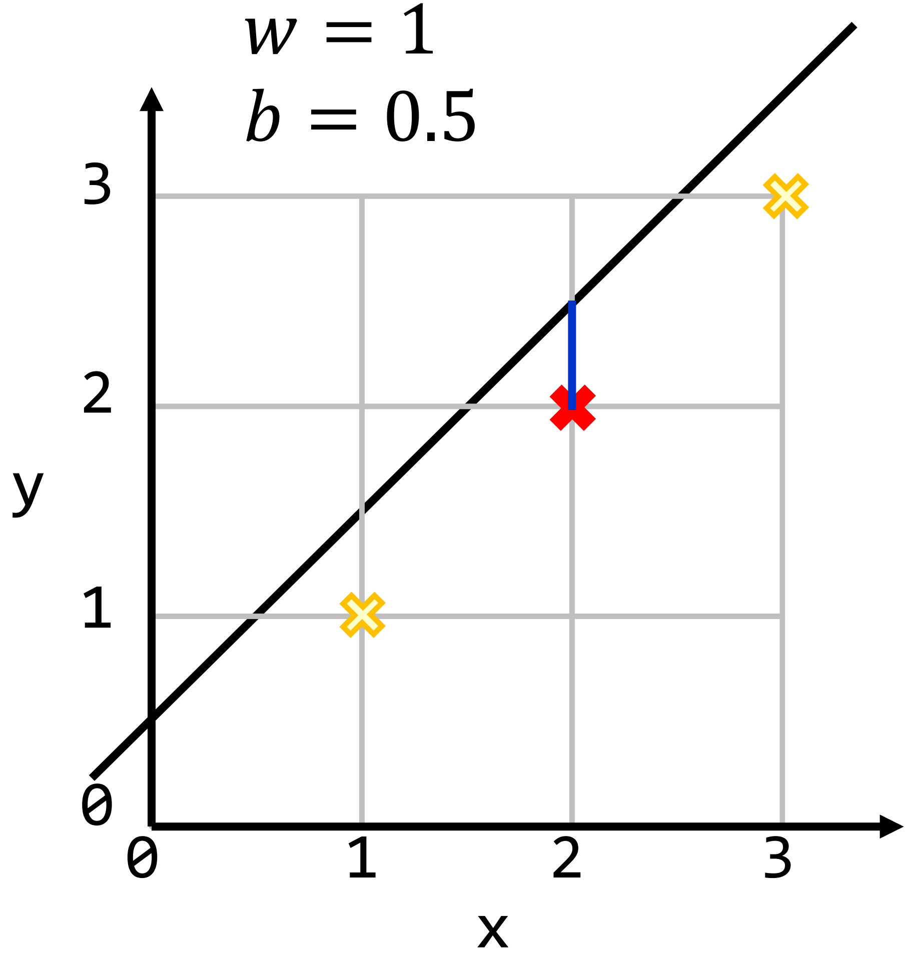
$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 0$$



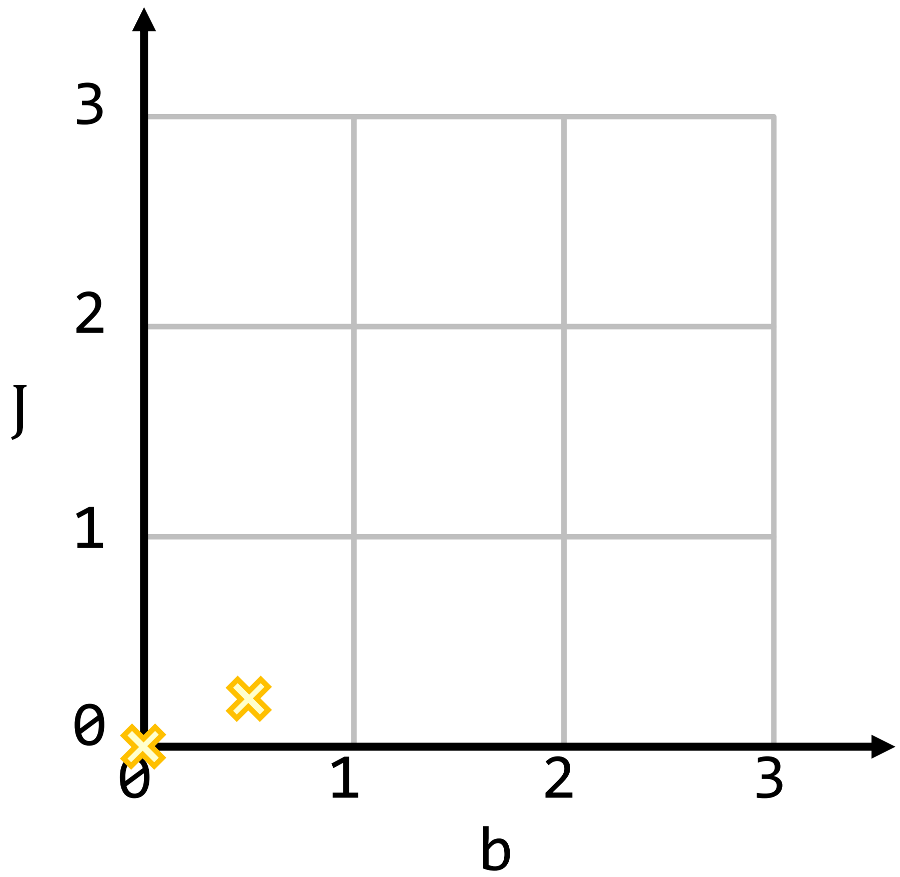
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



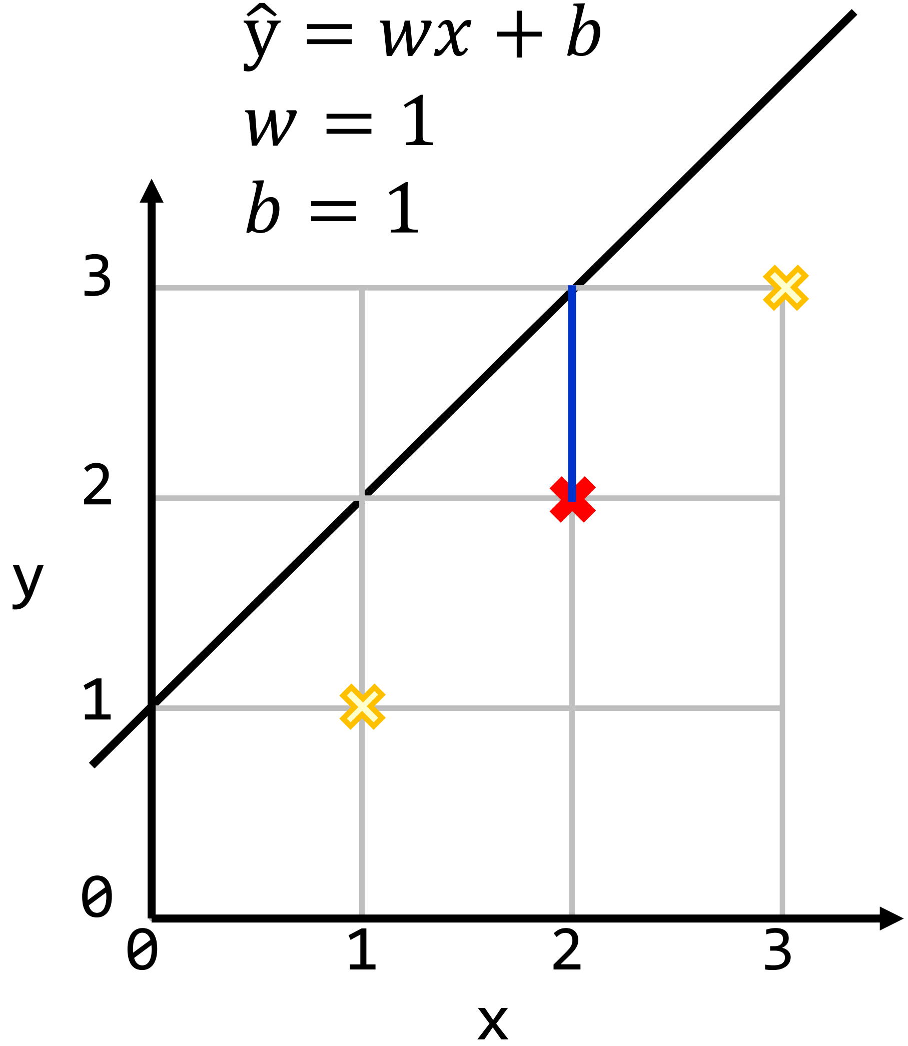
$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 0.5$$



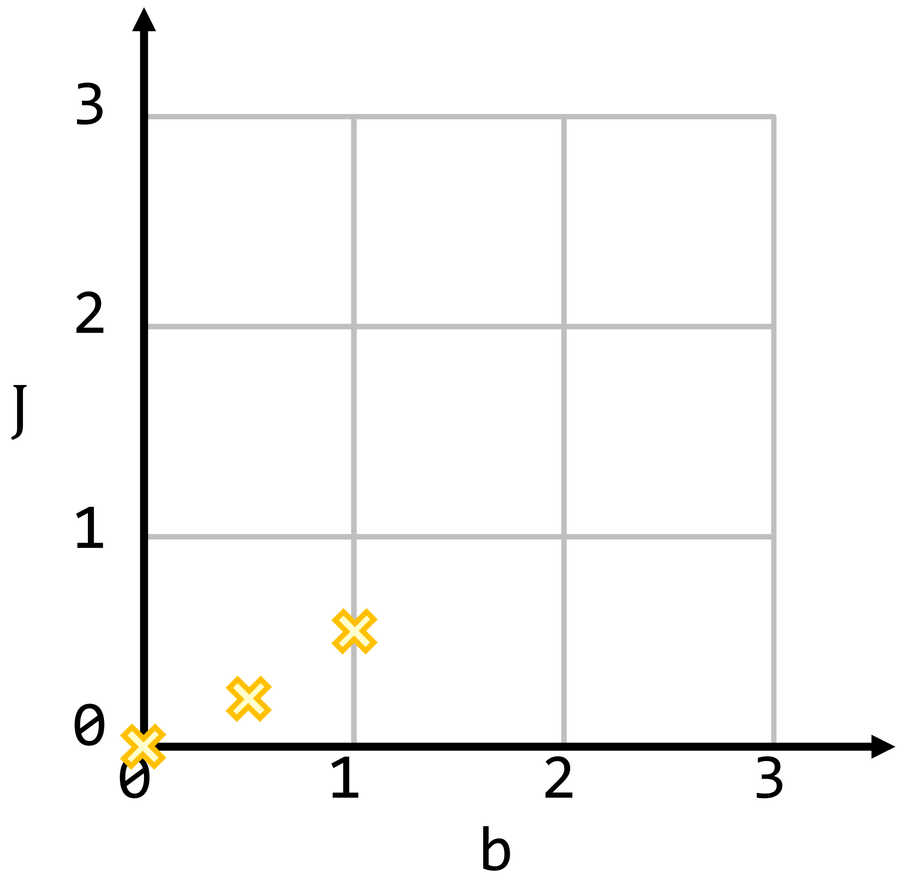
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



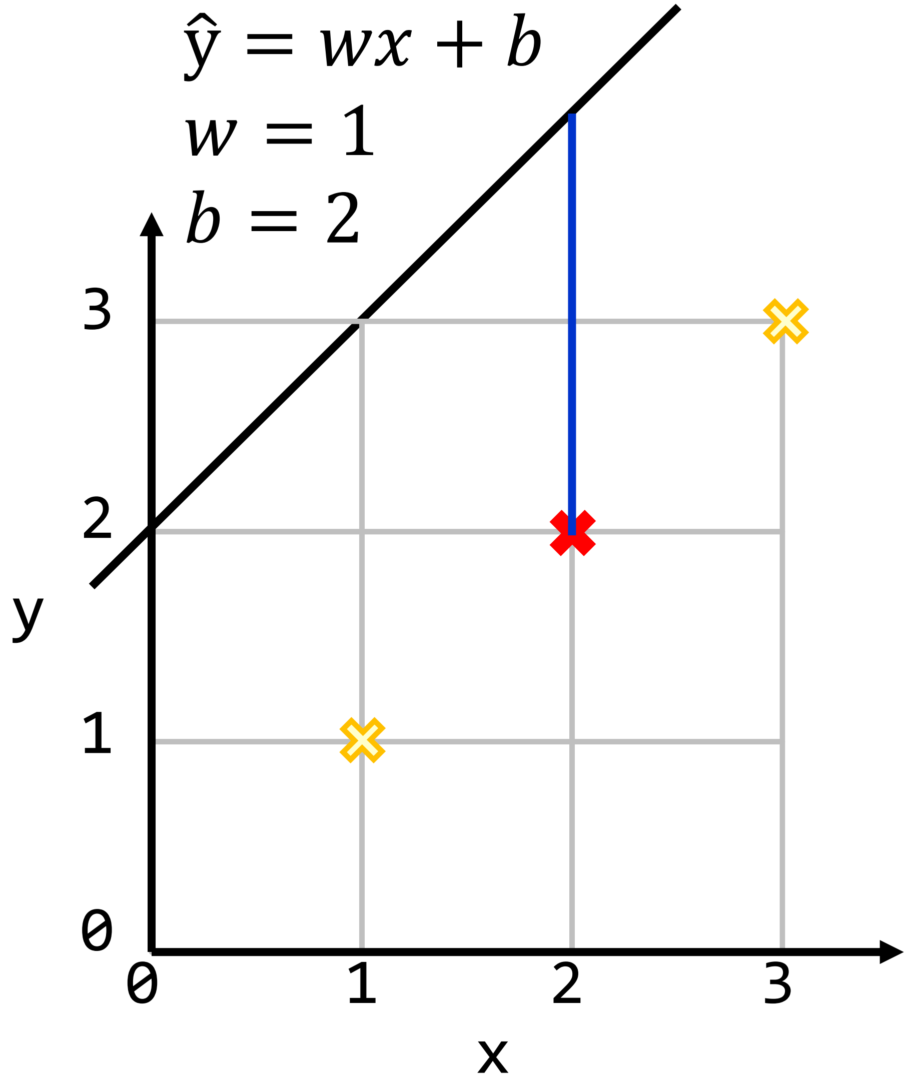
$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 1$$



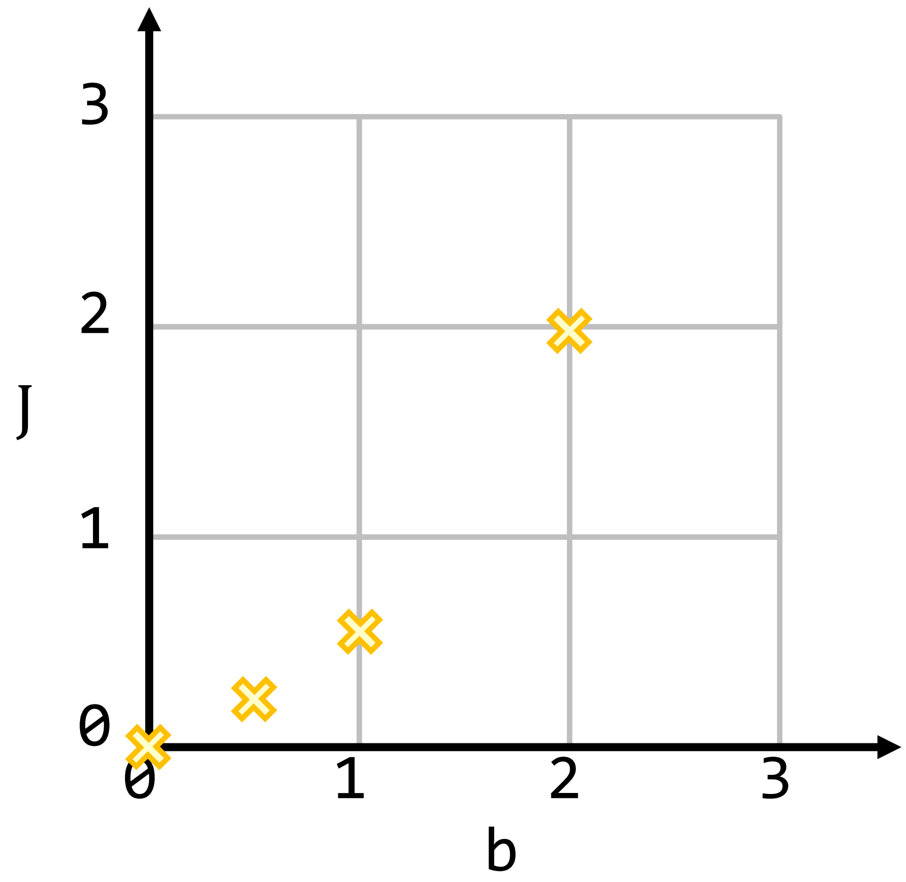
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 2$$

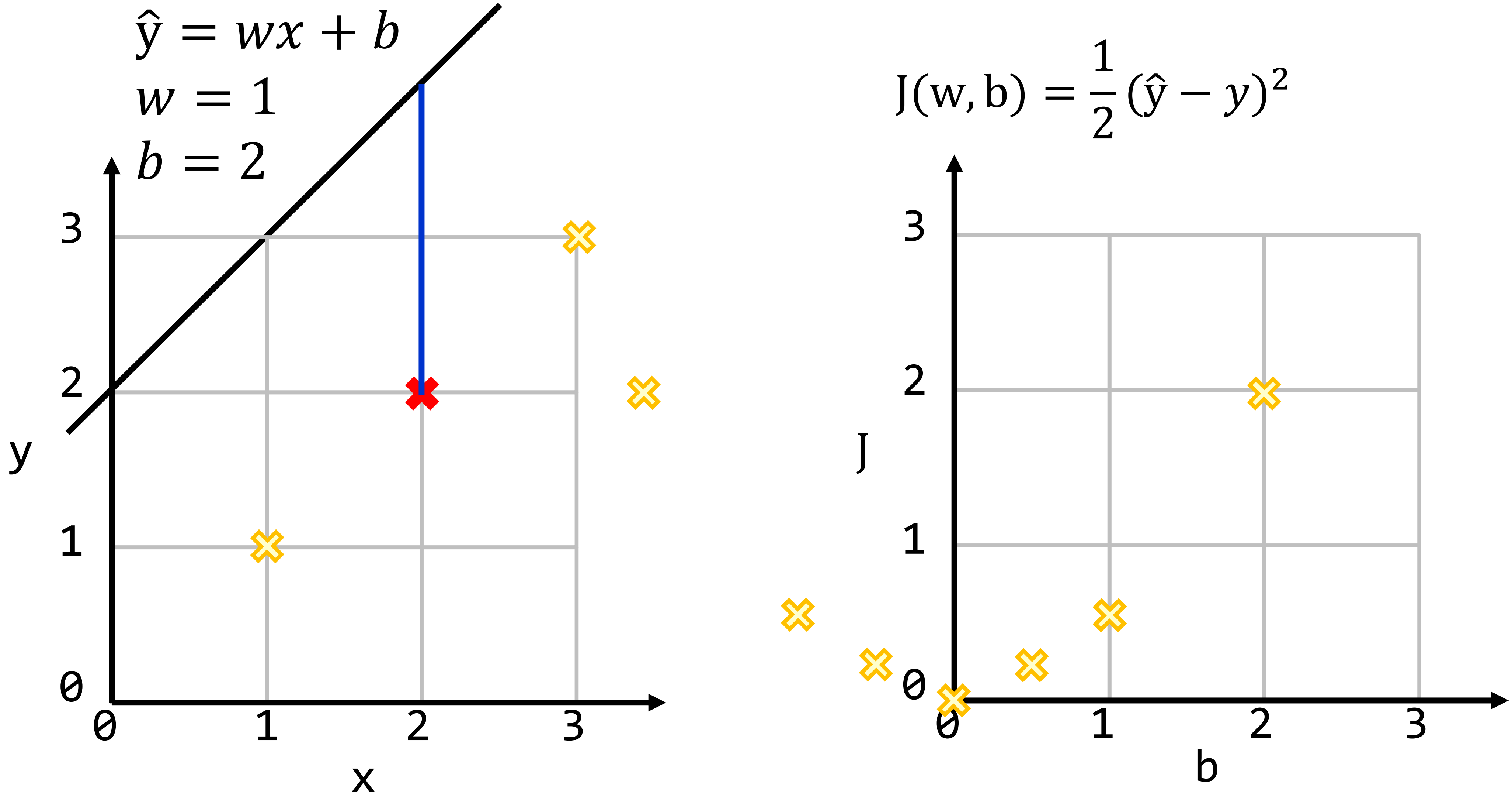


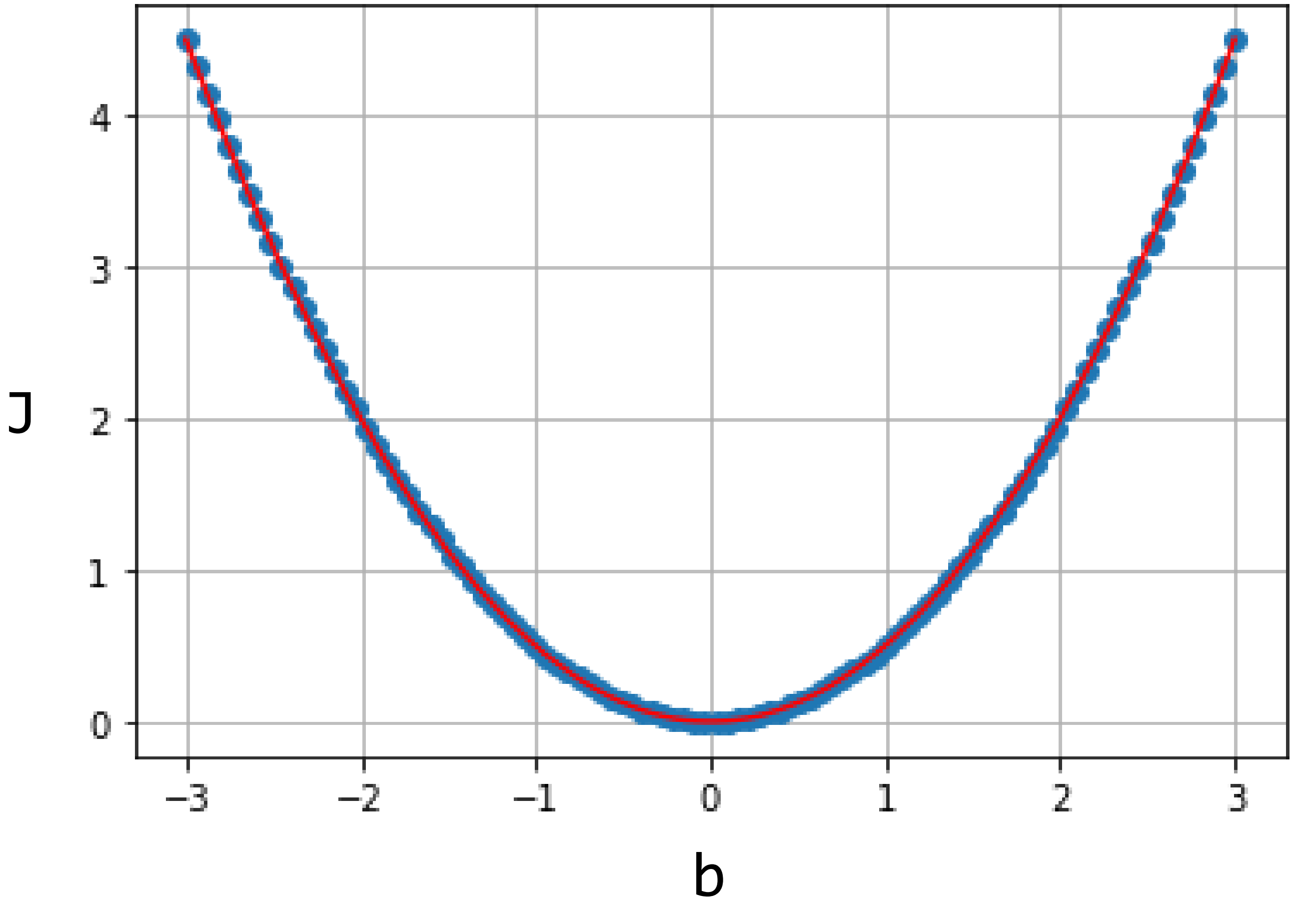
$$J(w,b) = \frac{1}{2}(\hat{y} - y)^2$$



$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 2$$

$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$




J
4
3
2
1
0
−3
−2
−1
0
1
2
3
b

2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

Gradient descent algorithm

- 비용 함수 최소화
- 경사 하강은 많은 최소화 문제에 사용된다.
- 주어진 비용 함수, 비용 (W, b)에 대해 비용을 최소화하기 위해 W, b를 찾는다.
- 일반적인 함수 : 비용 (w1, w2,...)에 적용 가능

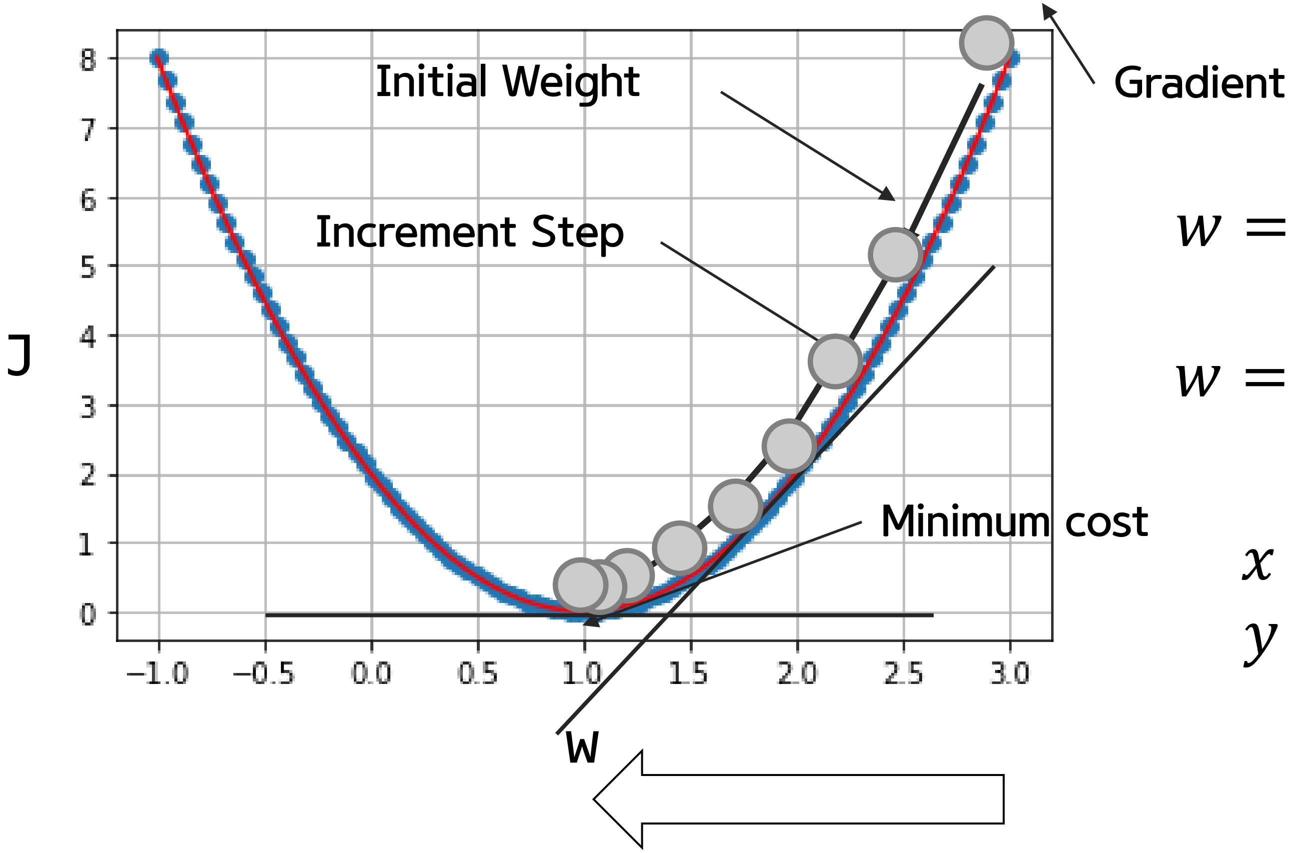
작동 방식

- 초기 추측으로 시작
  - 0,0 (또는 다른 값)에서 시작
  - W와 b를 약간 변경하여 cost(W, b)의 비용을 줄이려고 노력
- 매개 변수를 변경할 때마다 가능한 가장 낮은 cost(W, b)을 감소시키는 기울기를 선택
- 반복
- 최소한의 지역으로 수렴 할 때까지 수행

Gradient descent algorithm

$$\hat{y} = wx + b$$

$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



$$w = w \ - 기울기$$

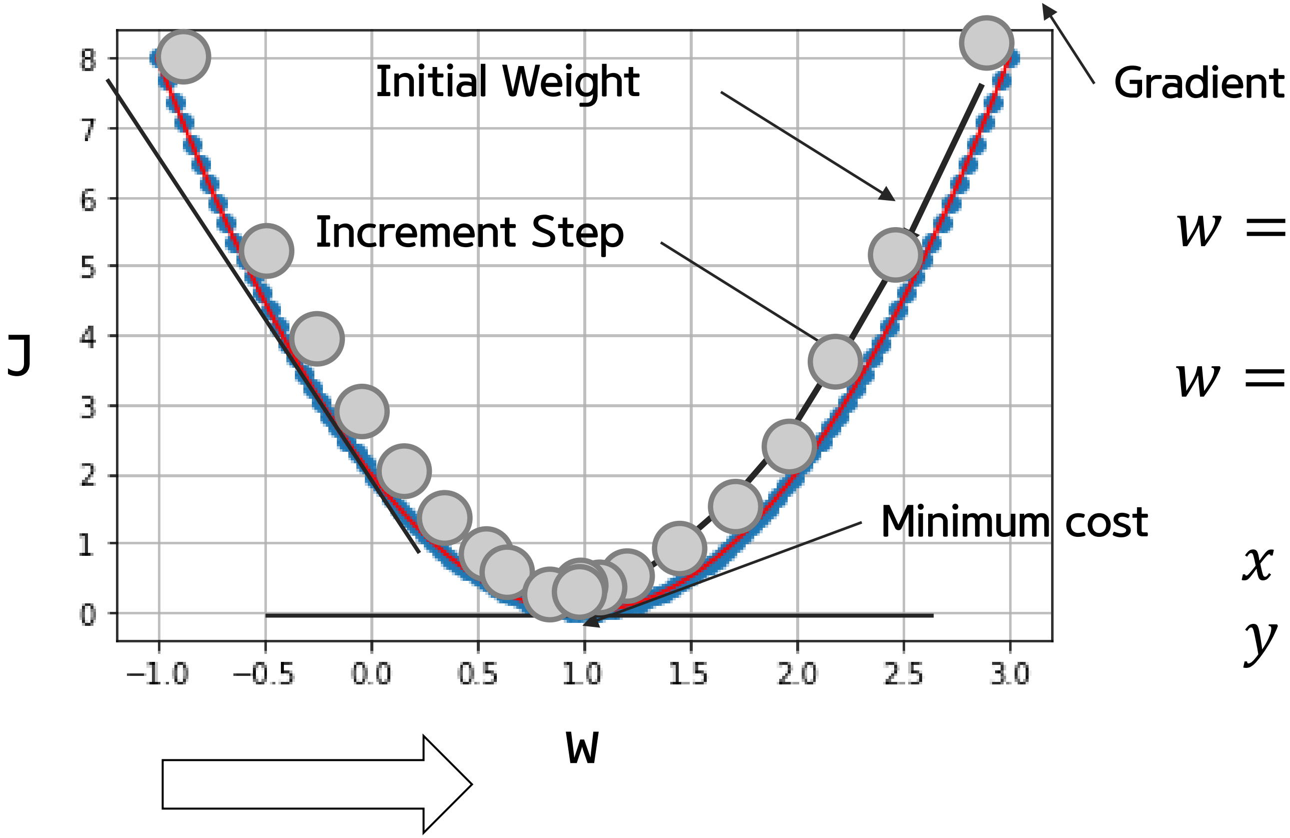
$$w = w \ - \alpha(\hat{y} - y)\mathrm{x}$$

$$x = 2$$
$$y = 2$$

Gradient descent algorithm

$$\hat{y} = wx + b$$

$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



$$w = w - 기울기$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

$$x = 2$$
$$y = 2$$

수렴까지 반복

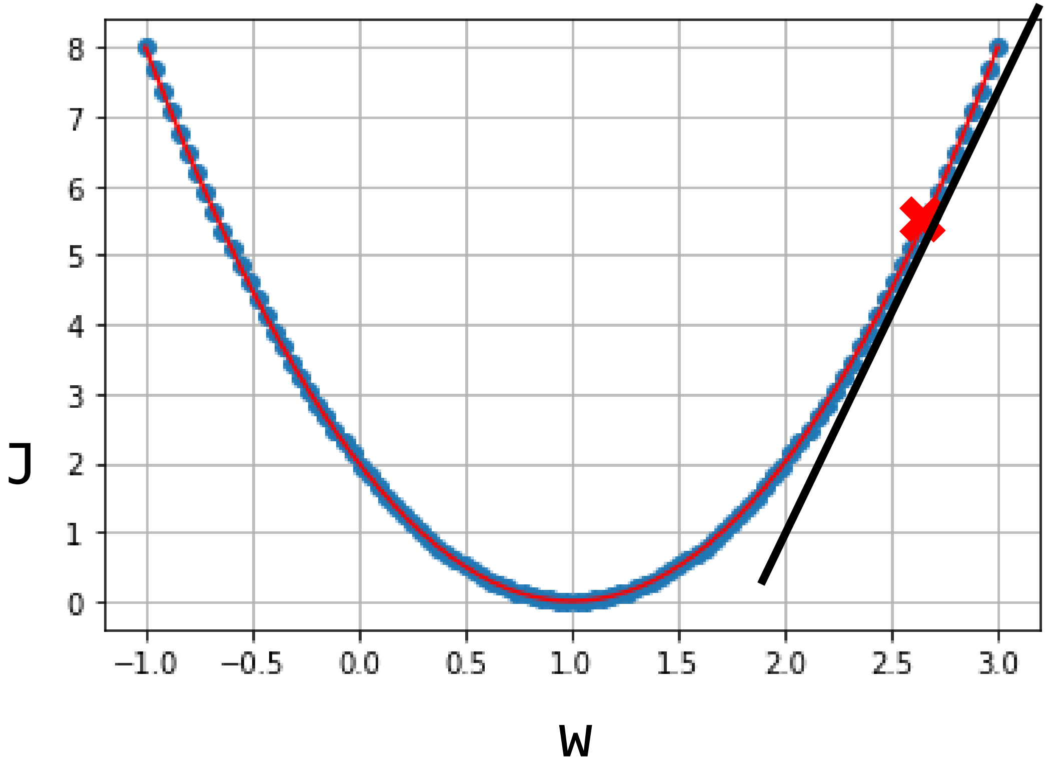
{

$$\mathrm{w} = \mathrm{w} - \alpha \frac{\partial}{\partial \mathrm{w}} J(w, b)$$

}

learning rate

derivative



$$\mathrm{J(w,b)} = \frac{1}{2}(\hat{\mathrm{y}} - \mathrm{y})^2$$

$\frac{\partial}{\partial \mathrm{w}} J(w, b)$는 $\frac{1}{2}(\hat{\mathrm{y}} - \mathrm{y})^2$ 을 w에 대해서 편미분 하면 된다.

$$\hat{y} = \mathrm{wx} + \mathrm{b}$$

$$\mathrm{J(w,b)} = \frac{1}{2}(\hat{y} - \mathrm{y})^2$$

$$= \frac{\partial}{\partial w}\frac{1}{2}((\mathrm{wx}+\mathrm{b}) - \mathrm{y})^2$$

$$= \frac{\partial}{\partial w}\frac{1}{2}((\mathrm{wx}+\mathrm{b})^2 - 2(\mathrm{wx}+\mathrm{b})\mathrm{y} + \mathrm{y}^2)$$

$$= \frac{\partial}{\partial w}\frac{1}{2}((\mathrm{wx}+\mathrm{b})^2 - 2\mathrm{ywx} - 2\mathrm{by} + \mathrm{y}^2)$$

$$= \frac{\partial}{\partial w}\frac{1}{2}(\mathrm{w}^2 x^2 + 2\mathrm{wxb} + \cancel{\mathrm{b}^2} - 2\mathrm{ywx} - \cancel{2\mathrm{by}} + \cancel{\mathrm{y}^2})$$

$$= \frac{1}{2}(2\mathrm{wx}^2 + 2\mathrm{xb} - 2\mathrm{yx})$$

$$= \mathrm{x}(\mathrm{wx} + \mathrm{b} - \mathrm{y})$$

$$= \mathrm{x}(\hat{y} - \mathrm{y})$$

$$\frac{\partial}{\partial \hat{y}} \frac{1}{2} (\hat{y} - y)^2 = \hat{y} - y$$

$$\frac{\partial}{\partial w} (wx + b) = x$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w} = x(\hat{y} - y)$$

Chain Rule 사용

$$\hat{y} = wx + b$$

$$J(w,b) = \frac{1}{2} (\hat{y} - y)^2$$

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w}$$

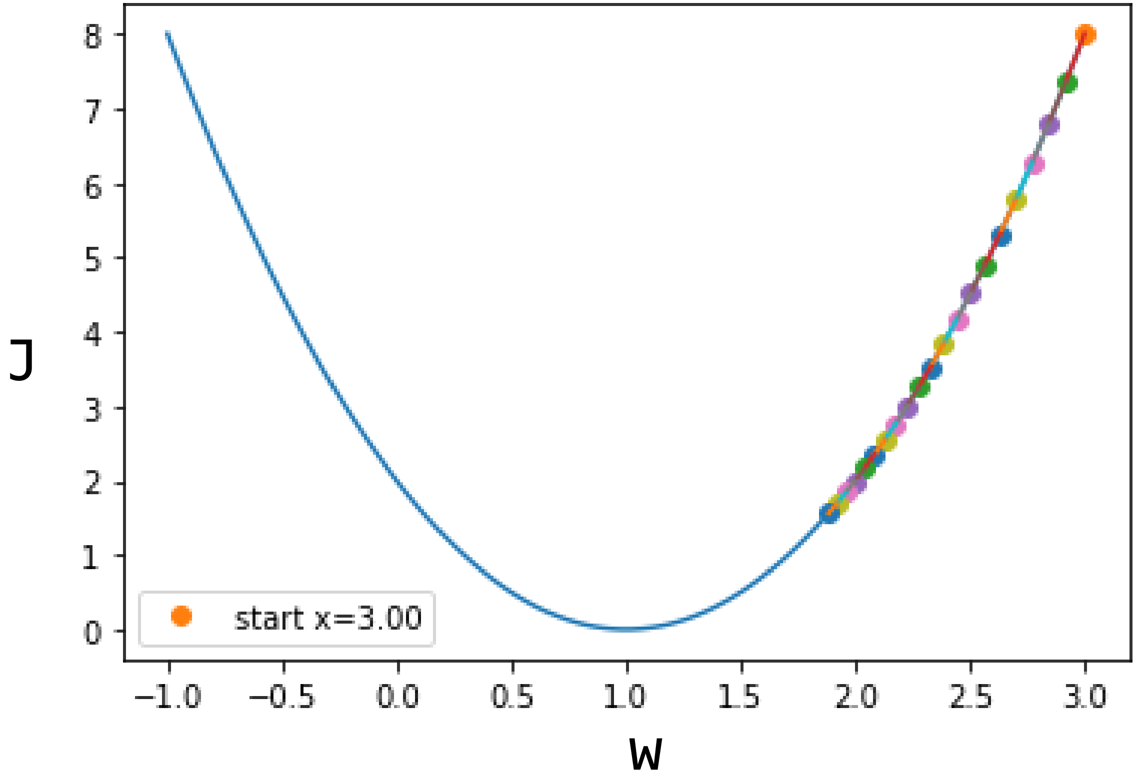
수렴까지 반복

{

$$w = w - \alpha * (\hat{y} - y) * x$$

}

α 가 0.01인 경우



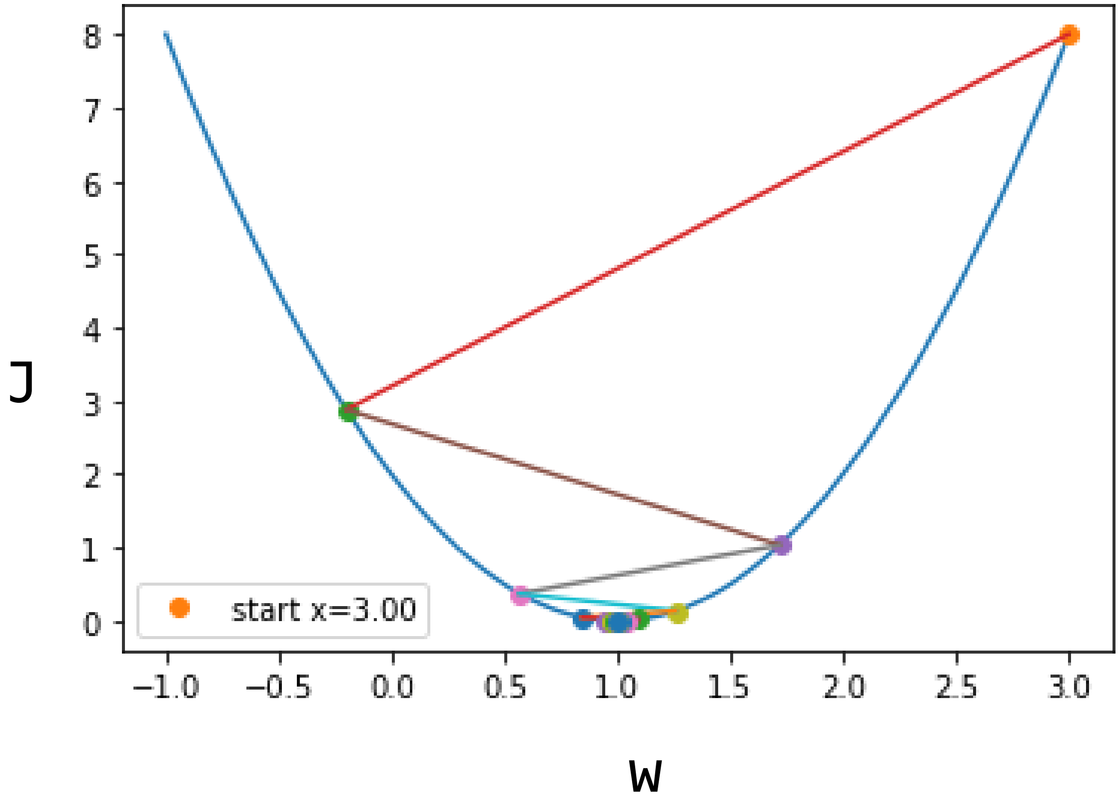
수렴까지 반복

{

$$w = w - \alpha * (\hat{y} - y) * x$$

}

α **가 0.4인 경우**

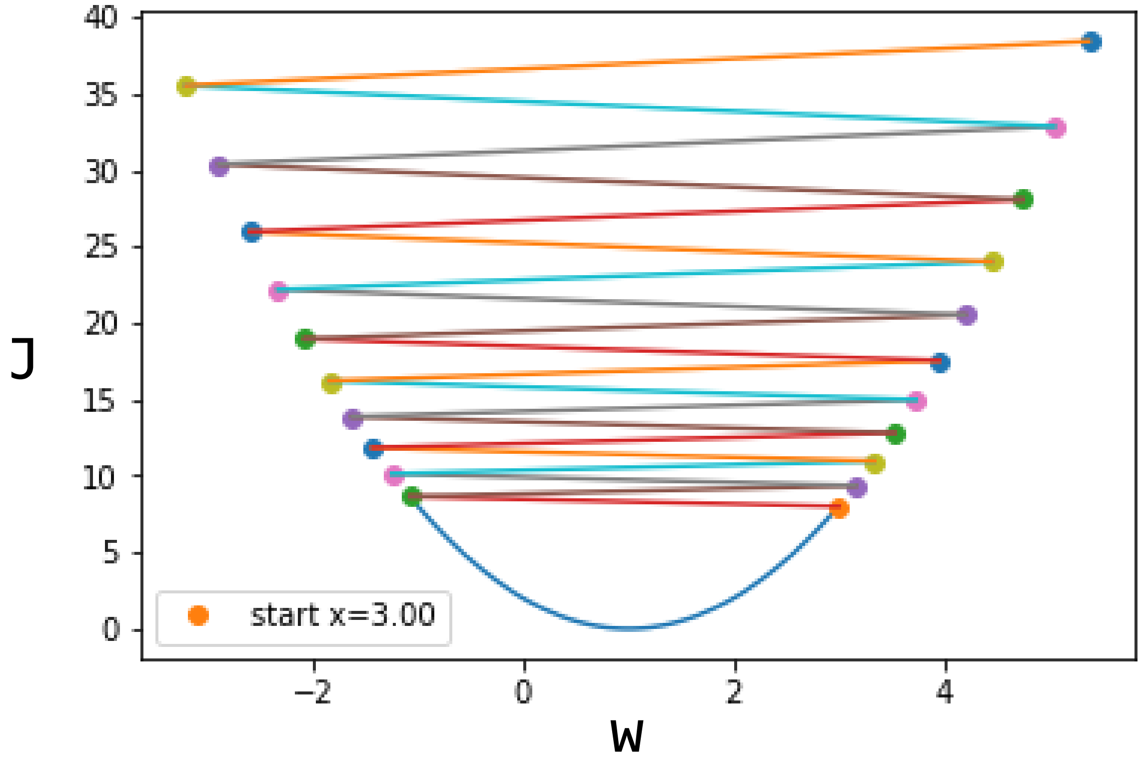


수렴까지 반복

{

$$w = w - \alpha * (\hat{y} - y) * x$$

}



α 가 0.51인 경우

$$\frac{\partial \mathrm{J}}{\partial \hat{\mathrm{y}}} = \frac{1}{2}(\hat{\mathrm{y}} - \mathrm{y})^2 = \hat{\mathrm{y}} - \mathrm{y}$$

$$\frac{\partial \hat{\mathrm{y}}}{\partial \mathrm{b}} = \mathrm{wx} + \mathrm{b} = 1$$

$$\frac{\partial \mathrm{J}}{\partial \hat{\mathrm{y}}} * \frac{\partial \hat{\mathrm{y}}}{\partial \mathrm{w}} = (\hat{\mathrm{y}} - \mathrm{y})$$

Chain Rule 사용

$$\hat{\mathrm{y}} = \mathrm{wx} + \mathrm{b}$$

$$\mathrm{J}(\mathrm{w,b}) = \frac{1}{2}(\hat{\mathrm{y}} - \mathrm{y})^2$$

$$\frac{\partial}{\partial \mathrm{b}} \mathrm{J}(\mathrm{w,b}) = \frac{\partial \mathrm{J}}{\partial \hat{\mathrm{y}}} * \frac{\partial \hat{\mathrm{y}}}{\partial \mathrm{b}}$$

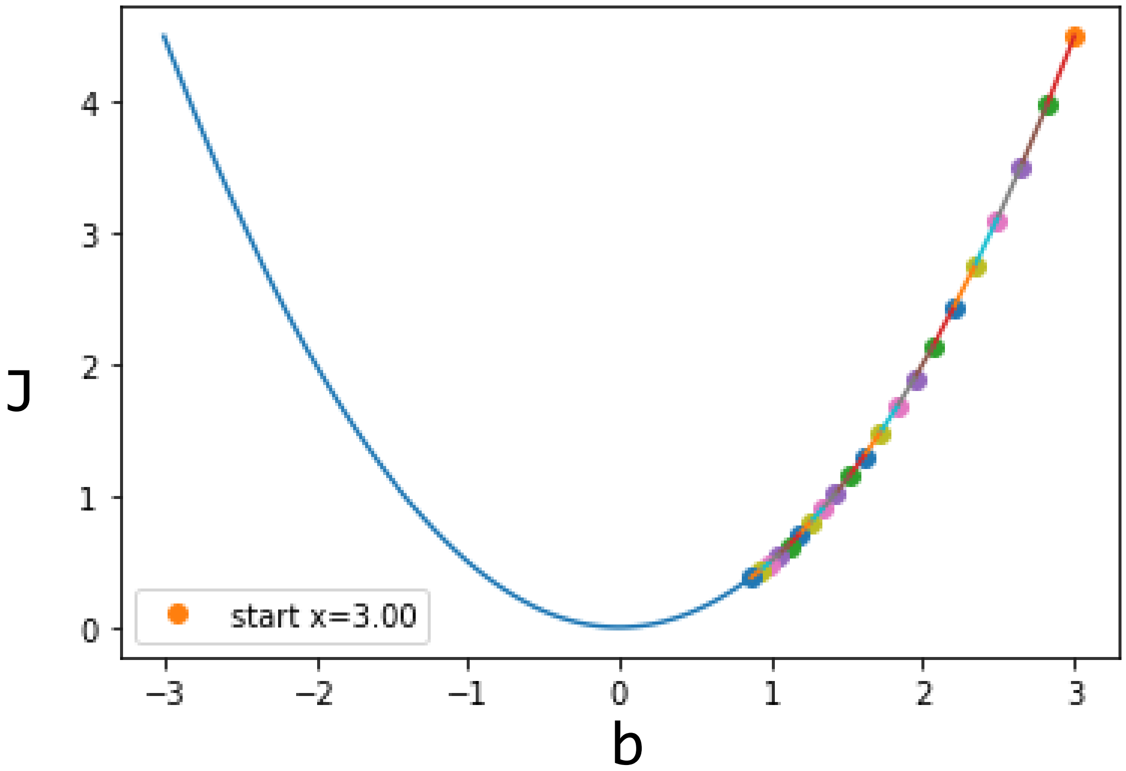
수렴까지 반복

{

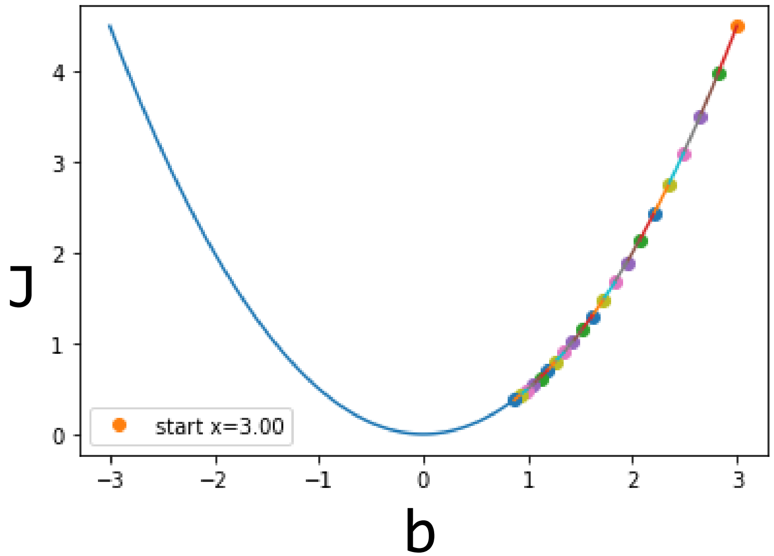
$$b = b - \alpha * (\hat{y} - y)$$

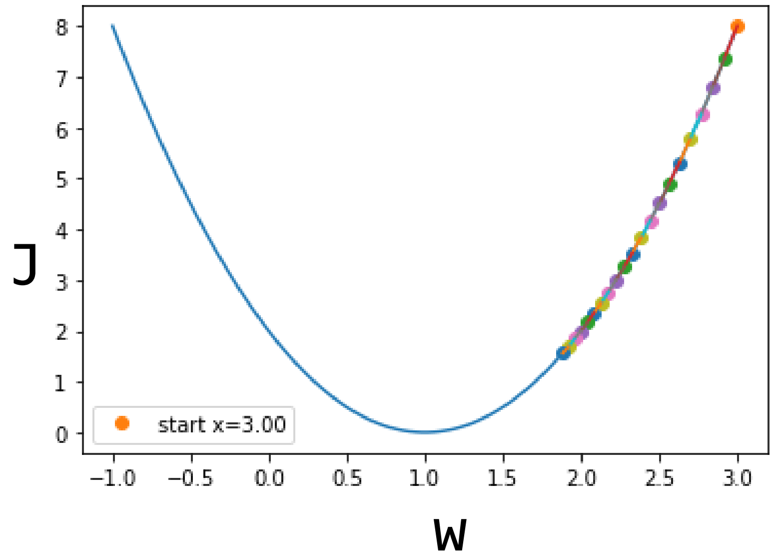
}

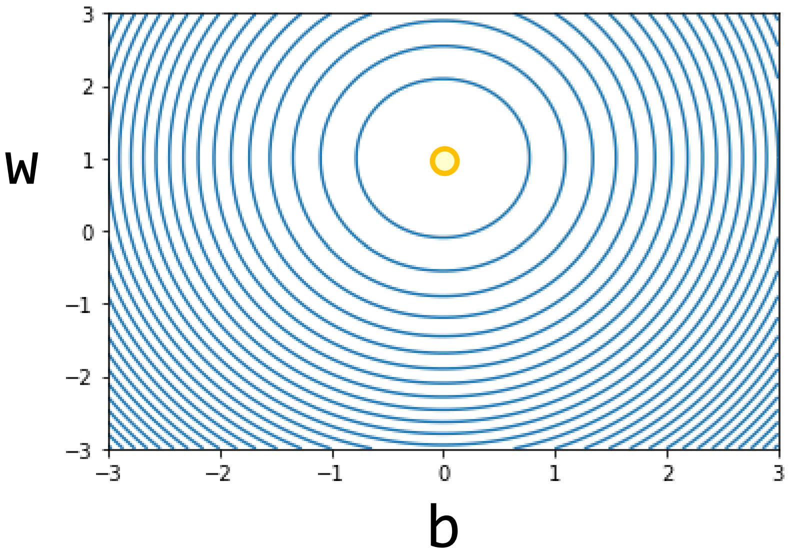
α **가 0.03인 경우**

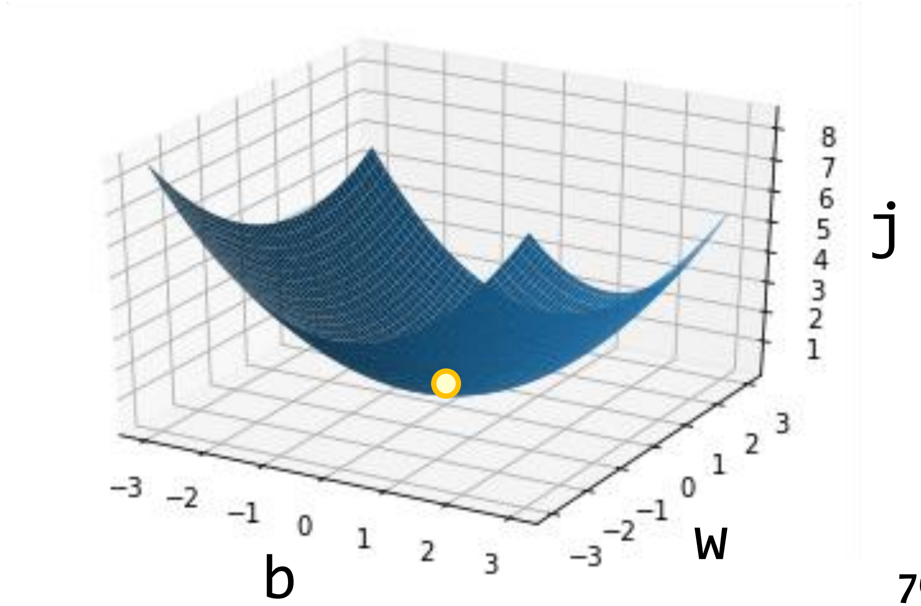


Gradient descent algorithm









2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

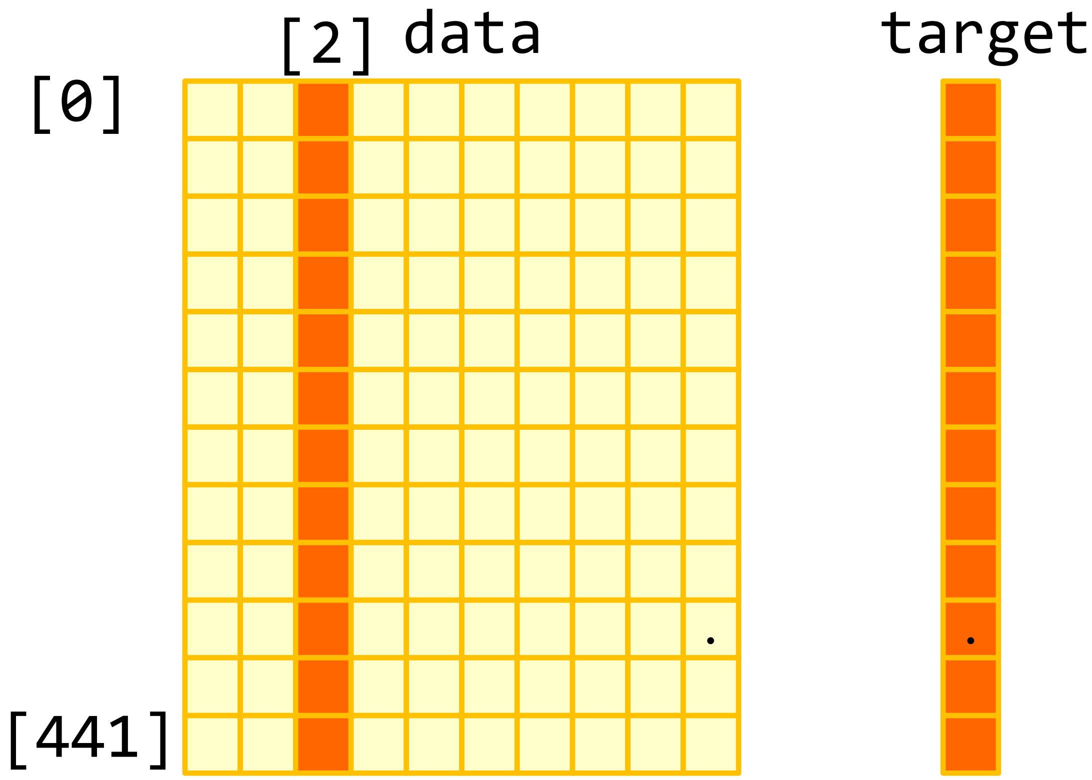
2.3 역전파 구현

2.4 선형회귀 구현

2.5 시그모이드 함수

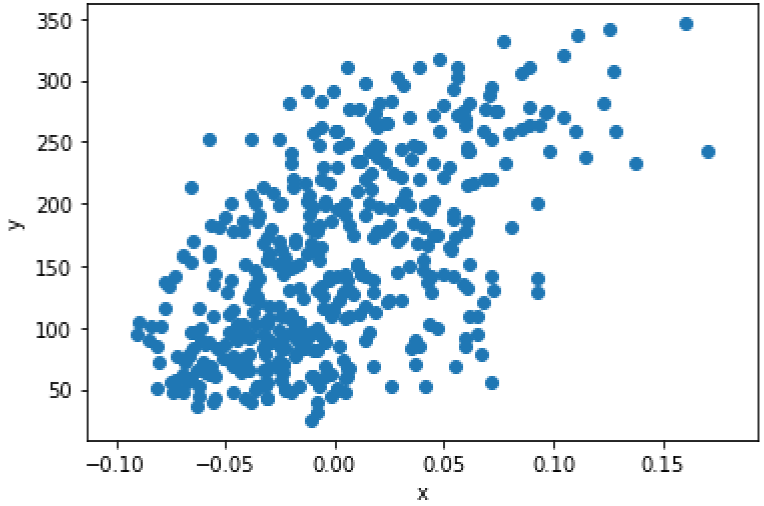
2.6 로지스틱 회귀 구현

data set 준비(sklearn 이용)



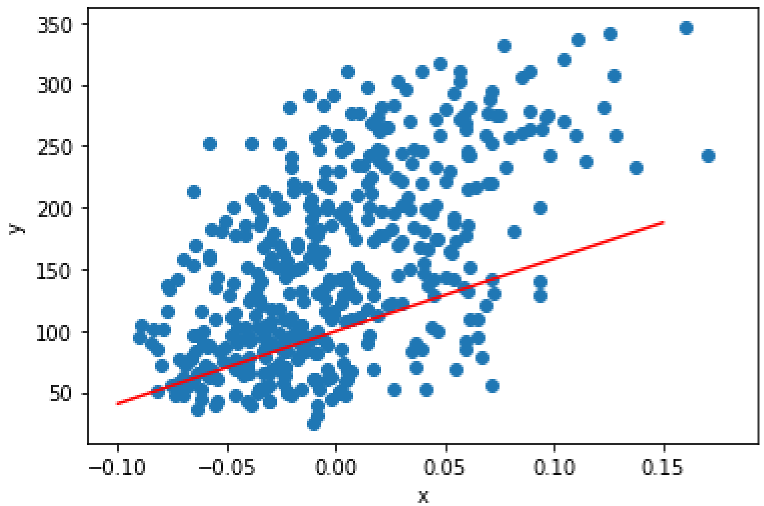
data set 준비(sklearn 이용)

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
import matplotlib.pyplot as plt

x = diabetes.data[:, 2]
y = diabetes.target

plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



역전파 구현
1. batch=1
2. 점의 전체 개수만큼 1번 순회 (epoch=1)
   : SGD=>원소 하나마다 w,b 갱신

```
w = 1.0
b = 1.0
rate = 0.01
for x_i, y_i in zip(x, y):
    y_hat = x_i * w + b
    err = y_hat - y_i
    w = w - rate * err * x_i
    b = b - rate * err

plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.15, 0.15 * w + b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]], 'r-')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

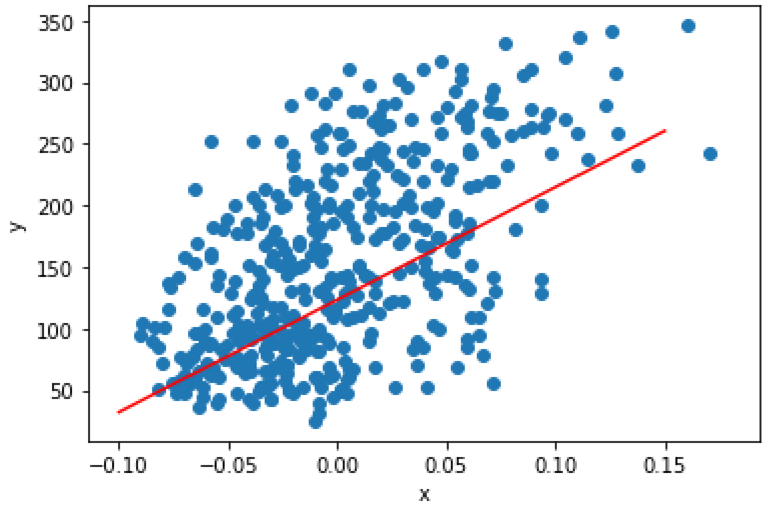


역전파 구현

1. batch=1
2. epoch=100번 만큼 외부 루프에서 순회
3. 점의 전체 개수만큼 1번 순회

   : SGD=>원소 하나마다 w,b 갱신

```
for i in range(1, 100):
    for x_i, y_i in zip(x, y):
        y_hat = x_i * w + b
        err = y_i - y_hat
        w_rate = x_i
        w = w + w_rate * err
        b = b + 1 * err
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.15, 0.15 * w + b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]], 'r-')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

Neuron class 만들기

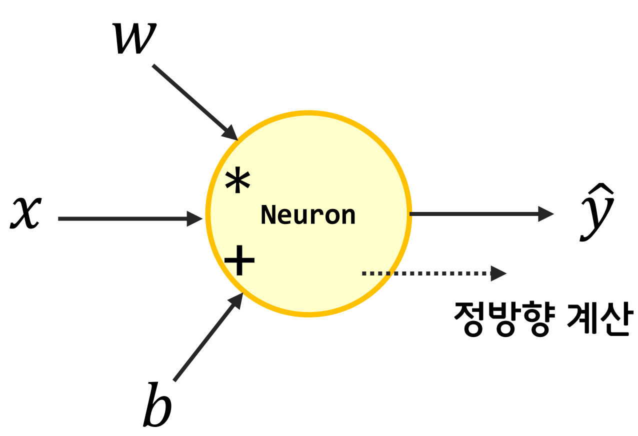
```
class Neuron:
    def __init__(self):
    # 초기화 작업을 수행합니다.
    ...

    # 필요한 함수를 추가합니다.
    ...
```

1. __init__() 함수 만들기

```
    def __init__(self):
        self.w = 1.0
        self.b = 1.0
```

2. 정방향 계산 만들기

```
    def forpass(self, x):
        y_hat = x * self.w + self.b       # 직선 방정식을 계산합니다
        return y_hat
```



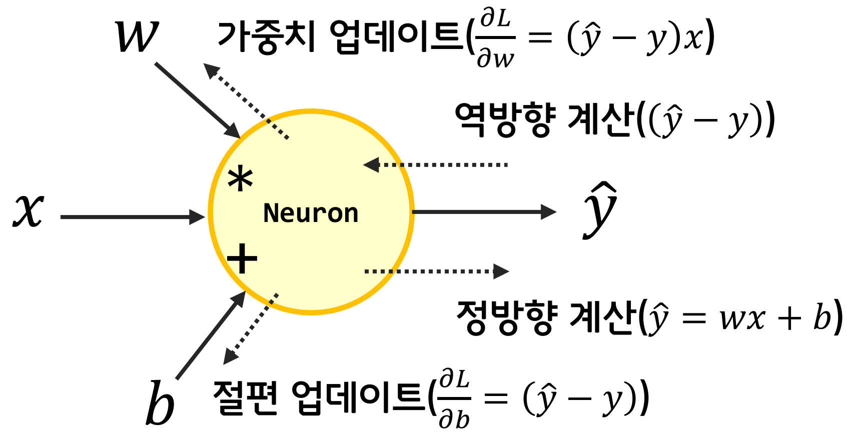
$$\hat{y} = wx + b$$

3. 역방향 계산 만들기

```
    def backprop(self, x, err):
        w_grad = x * err    # 가중치에 대한 그래디언트를 계산합니다
        b_grad = 1 * err    # 절편에 대한 그래디언트를 계산합니다
        return w_grad, b_grad
```

$$\frac{\partial L}{\partial w} = (\hat{y} - y)x$$

$$\frac{\partial L}{\partial b} = (\hat{y} - y)$$



4. 훈련을 위한 fit() 함수 구현

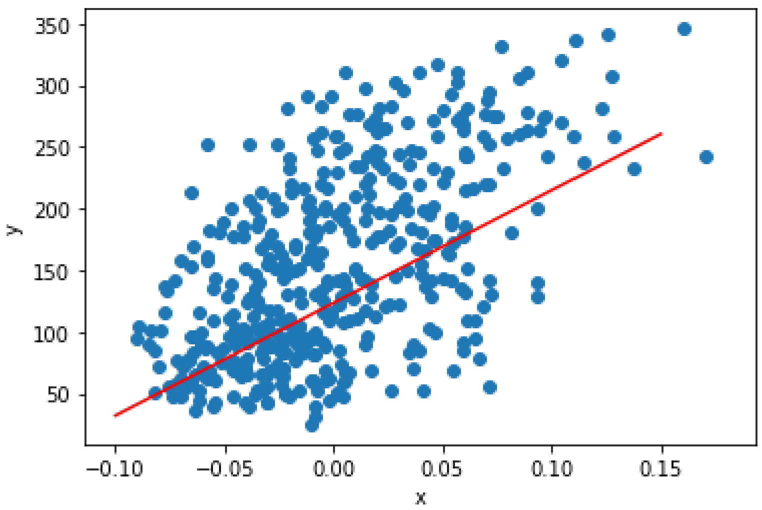
```
    def fit(self, x, y, epochs=100, rate=0.01):
        for i in range(epochs):            # 에포크만큼 반복합니다
            for x_i, y_i in zip(x, y):    # 모든 샘플에 대해 반복합니다
                y_hat = self.forpass(x_i) # 정방향 계산
                err = y_hat - y_i       # 오차 계산
                w_grad, b_grad = self.backprop(x_i, err)  # 역방향 계산
                self.w -= rate*w_grad          # 가중치 업데이트
                self.b -= rate*b_grad          # 절편 업데이트
```

5. 모델 훈련하기

```
neuron = Neuron()
neuron.fit(x, y)
```

6. 학습이 완료된 모델의 가중치와 절편 확인

```
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * neuron.w + neuron.b)
pt2 = (0.15, 0.15 * neuron.w + neuron.b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]],'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

Logistic Regression

- 로지스틱 회귀 란?
  - Classification(분류)
  - Logistic vs Linear

- 동작 방식
  - 가설 표현
  - Sigmoid 함수
  - Decision Boundary(결정경계)
  - Cost Function
  - Optimizer (Gradient Descent)

Classification

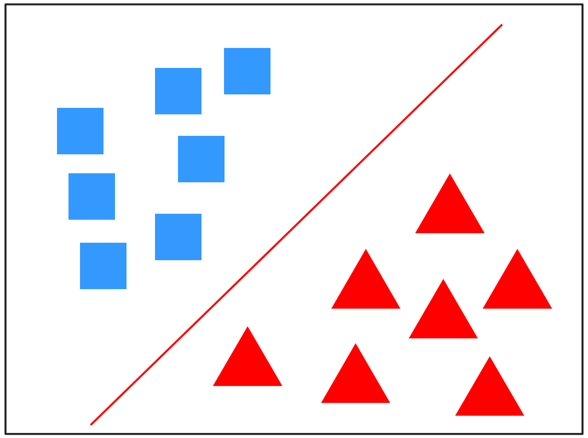
Binary Classification(이진 분류) 란?

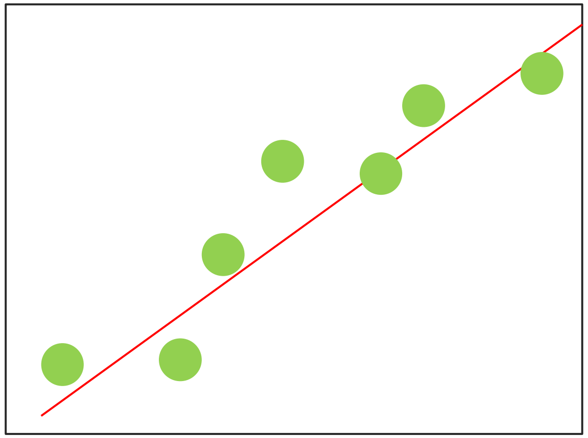
: 값은 0 또는 1

- Exam : Pass or Fail
- Spam : Not Spam or Spam
- Face : Real or Fake
- Tumor : Not Malignant or Malignant

Logistic vs Linear

로지스틱 회귀와 선형 회귀의 차이점은?





Discrete(분리) : 분류 목적

신발 사이즈 / 회사의 근로자수

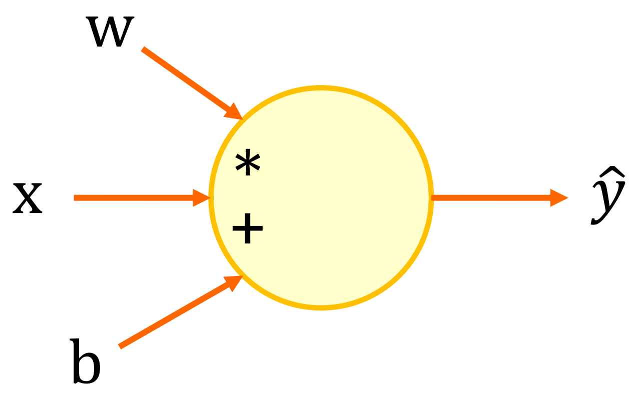
Continuous (지속적인) : 값의 예측

시간/ 무게 / 높이

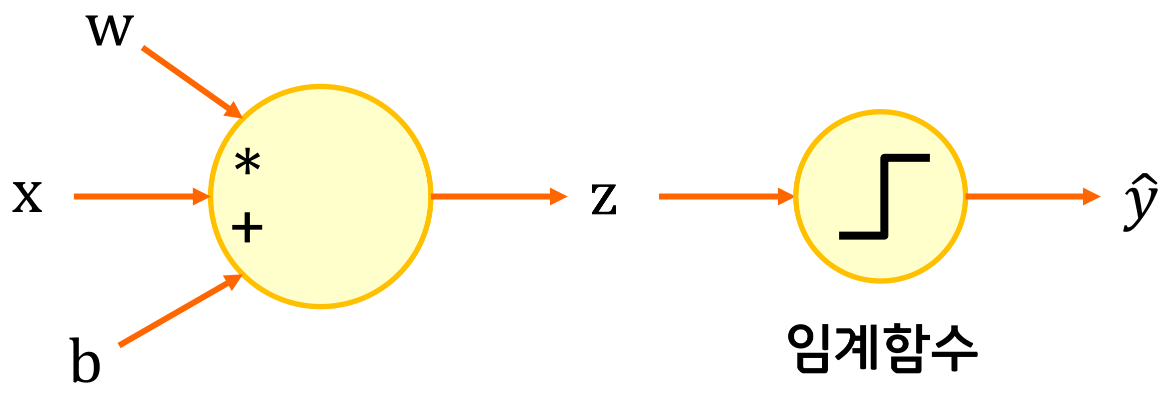
keras로 구현한 이진 분류 문제

```
from keras.models import Sequential # 케라스의 Sequential()을 임포트
from keras.layers import Dense      # 케라스의 Dense()를 임포트
from keras import optimizers        # 케라스의 옵티마이저를 임포트
import numpy as np # Numpy를 임포트
import matplotlib.pyplot as plt

X=np.array([1,2,3,4,5,6,7,8])
Y=np.array([0,0,0,0,1,1,1,1])

model=Sequential()
model.add(Dense(1, input_dim=1, activation='sigmoid'))
RMSprop=optimizers.RMSprop(lr=0.01)
model.compile(optimizer=RMSprop ,loss='binary_crossentropy',metrics=['ac
curacy'])
model.fit(X,Y, batch_size=1, epochs=200, shuffle=False)

plt.plot(X, Y, 'rx')
print(X)
print(model.predict(X))
plt.show()
```

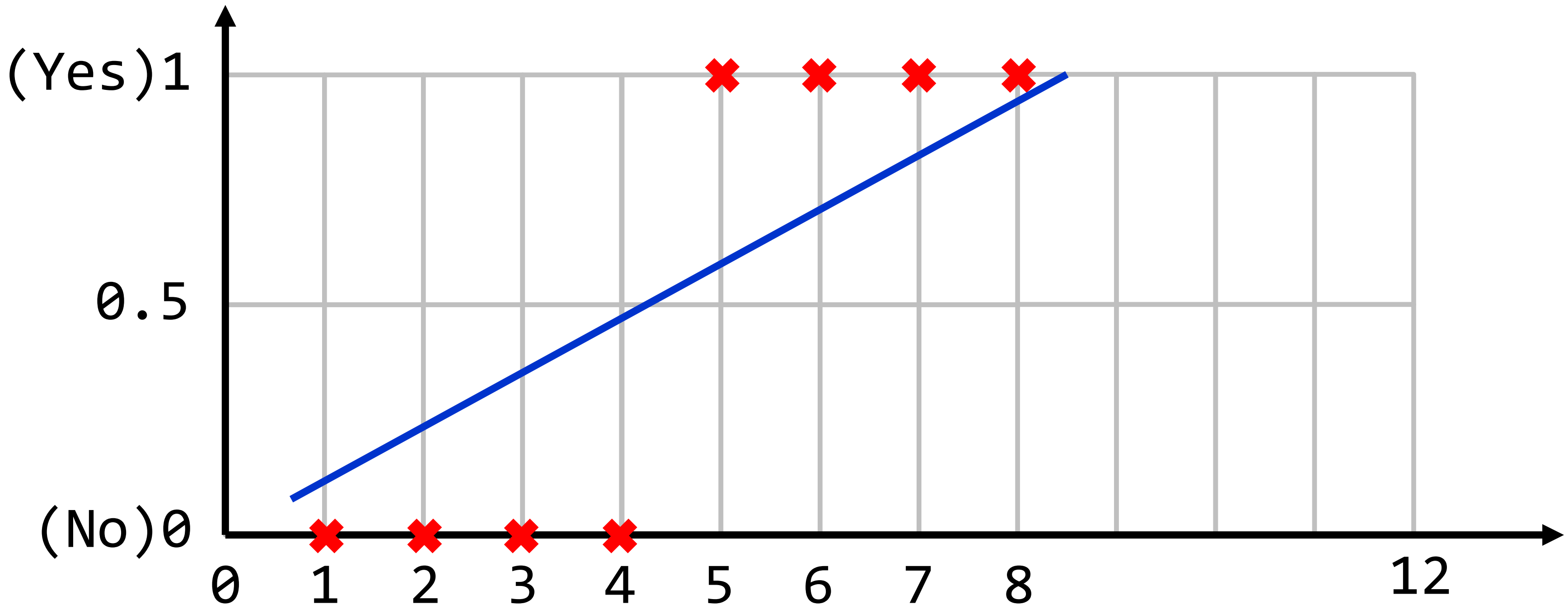
퍼셉트론



퍼셉트론



Sigmoid 함수 사용 이유

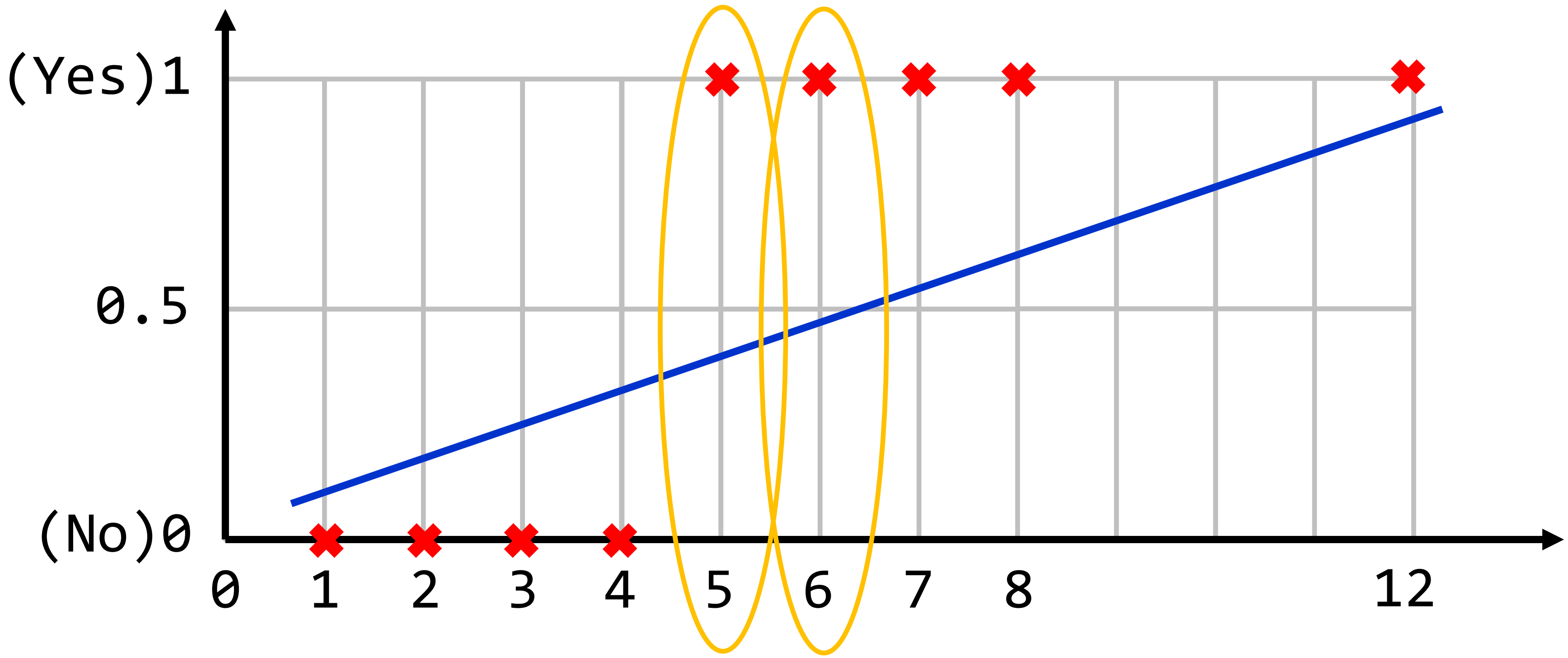


Threshold classifier output at 0.5:

If $\hat{y} \geq 0.5$, predict "y=1"
If $\hat{y} < 0.5$, predict "y=0"

1 => 양성 5 => 악성
2 => 양성 6 => 악성
3 => 양성 7 => 악성
4 => 양성 8 => 악성

Sigmoid 함수 사용 이유

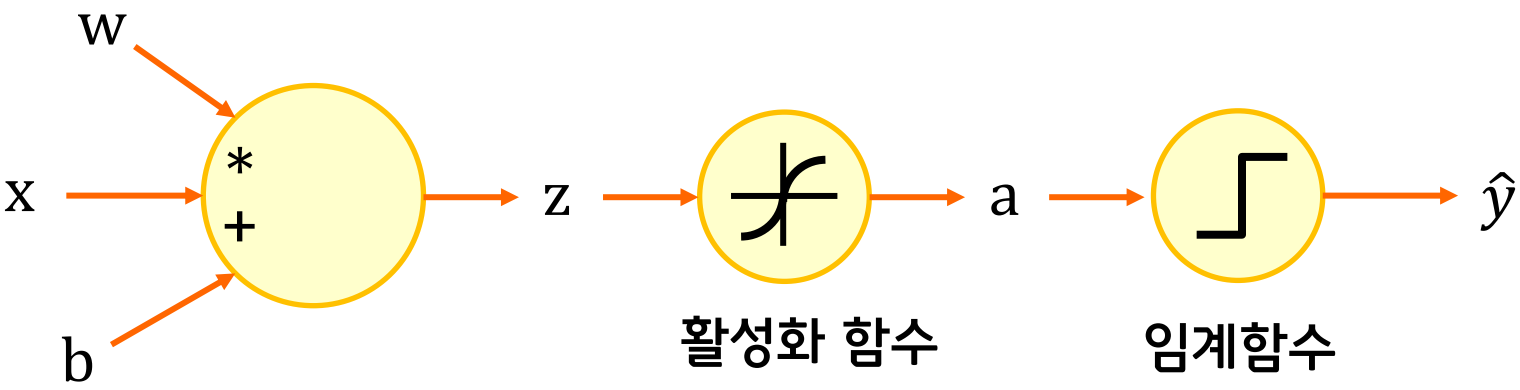


Threshold classifier output at 0.5:

If $\hat{y} \geq 0.5$, predict "y=1"
If $\hat{y} < 0.5$, predict "y=0"

1 => 양성
2 => 양성
3 => 양성
4 => 양성
5 => 양성
6 => 양성
7 => 악성
8 => 악성

Sigmoid 함수 사용 이유



Sigmoid 함수 사용 이유

Classification: y=0 or 1

$\hat{y}$ can be > 1 or < 0

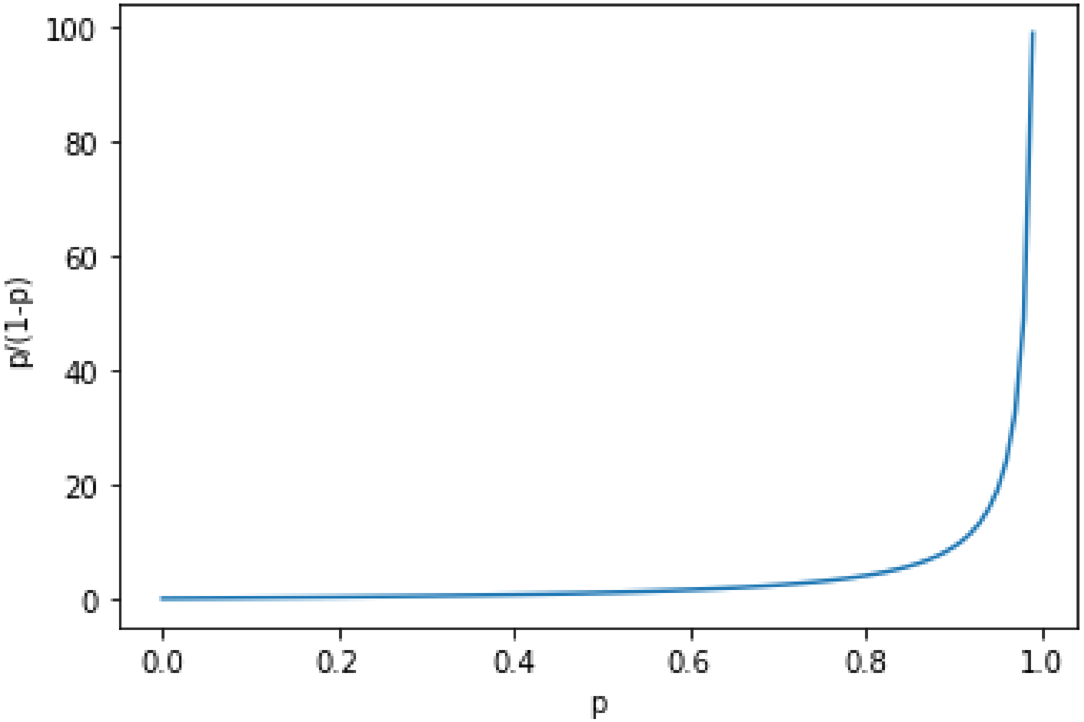
Logistic Regression: $0 < \hat{y} < 1$

분류 문제 사용

오즈 비(odds ratio)

$$\text{OR(odds ratio)} = \frac{p}{1-p} \quad (p=\textbf{성공확률})$$

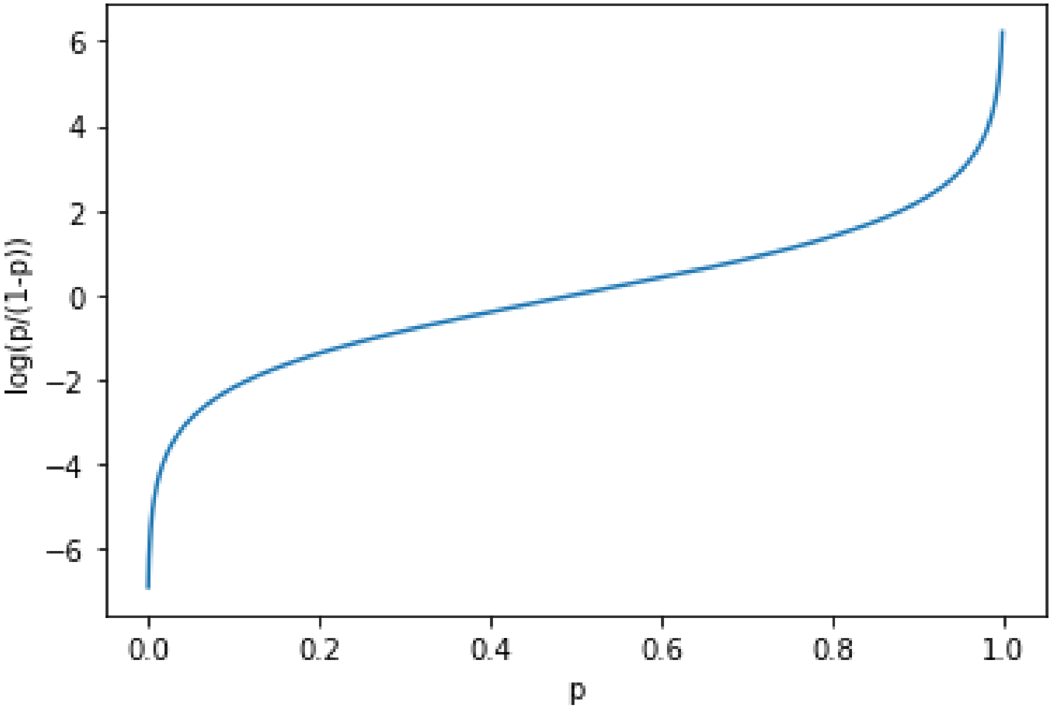
오즈 비를 그래프로 그리면 다음과 같다.
p가 0부터 1까지 증가할 때 오즈 비의 값은 처음에는
천천히 증가하지만 p가 1에 가까워지면 급격히 증가한다.



로짓 함수(Logit function)

$$\text{logit(p)} = \log(\frac{p}{1-p})$$ (p=**성공확률**)

로짓 함수는 p가 0.5일 때 0이 되고 p가 0과 1일 때 각각 무한대로 음수와 양수가 되는 특징을 가진다.



시그모이드 함수(Sigmoid function)

$$\mathrm{logit(p)} = \log(\frac{p}{1-p}) = z$$

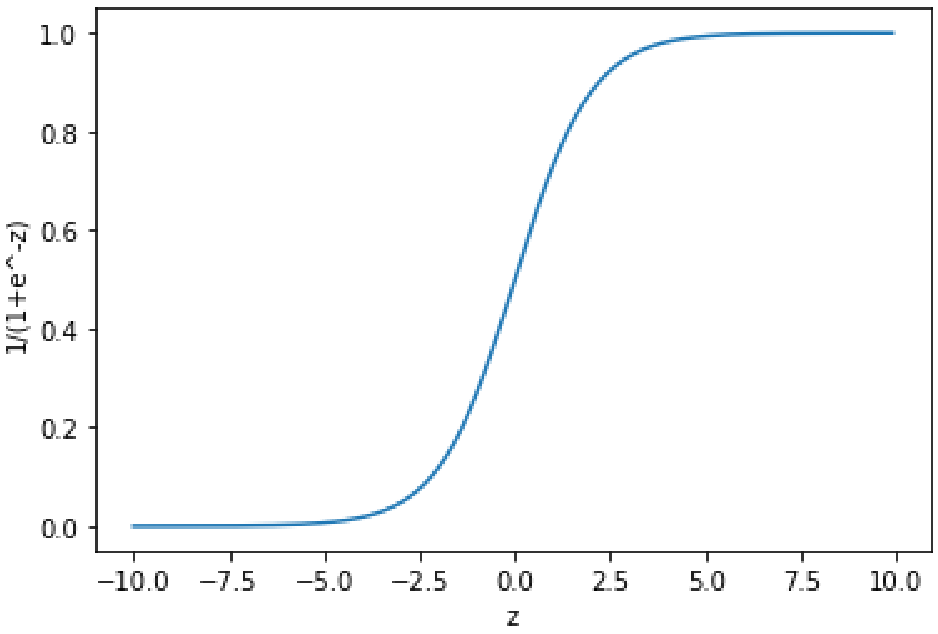
로지틱 함수의 유도 : p에대해 정리

$$\log(\frac{p}{1-p}) = z$$

$$e^{\log(\frac{p}{1-p})} = e^{z}$$

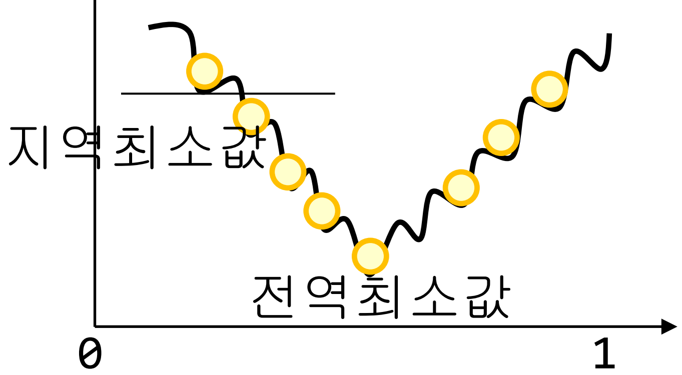
$$\frac{p}{1-p} = e^{z}$$

$$\mathrm{p} = (1-p) * e^{z}$$
$$\mathrm{p} = e^{z} \ - p \ * e^{z}$$
$$\mathrm{p} + \mathrm{p}* e^{z} = e^{z}$$
$$\mathrm{p} = \frac{e^{z}}{1+e^{z}} = \frac{1}{\frac{1}{e^{z}}+1} = \frac{1}{1+e^{-z}}$$

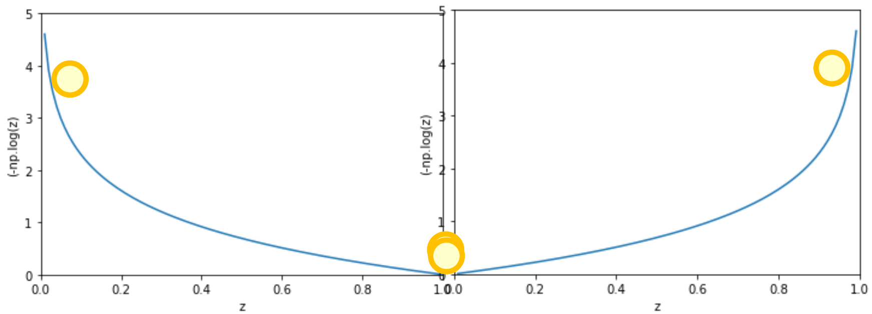


A convex logistic regression cost function

로지스틱 회귀에서는 MSE를 사용할 수 없다.
이유 : 지역 최솟값 생성 때문



Binary Cross Entropy 함수



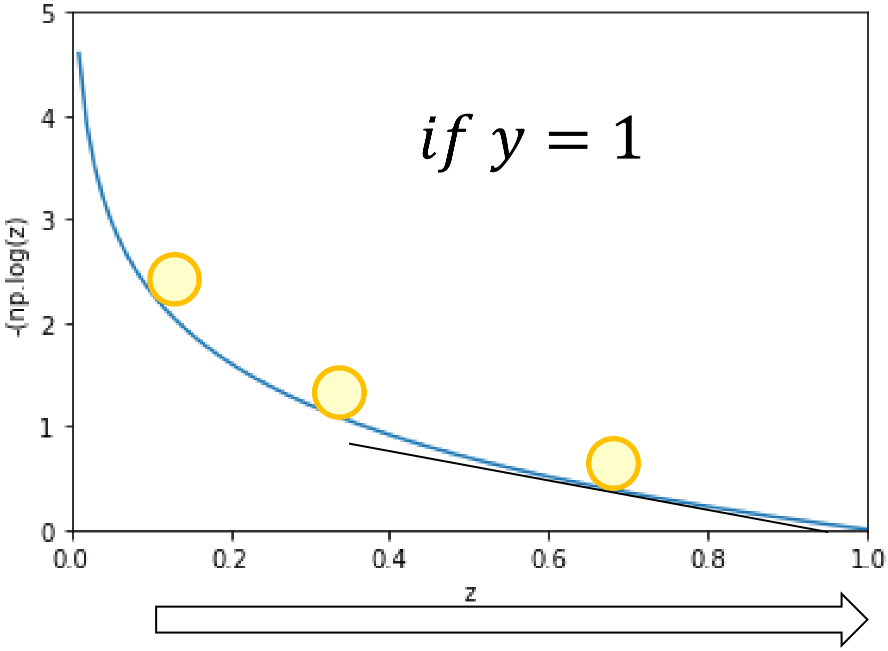
$$\mathrm{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$

$$J(w) = \frac{1}{2}(sigmoid(\hat{y}) - y)^2$$

$$\mathrm{cost}(\hat{y}, y) = -(\mathrm{y} * \log(a) + (1-y)\log(1-\mathrm{a}))$$

Logistic regression cost function

$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$



$\text{Cost} = 0 \text{ if } y = 1,\ \hat{y} = 1$

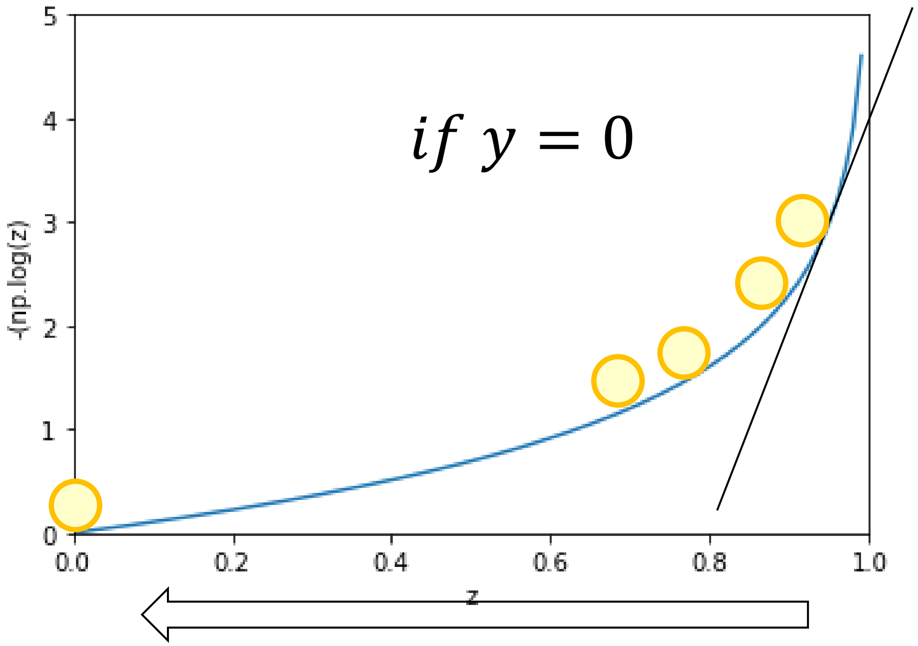
$\text{But as } \hat{y} \rightarrow 0$

$\text{Cost} \rightarrow \infty$

로지스틱 회귀 손실 함수

Logistic regression cost function

$$\mathrm{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$



$\mathrm{Cost} = 0$ if $y = 0,\ \hat{y} = 0$

But as $\hat{y} \rightarrow 1$

$\mathrm{Cost} \rightarrow \infty$

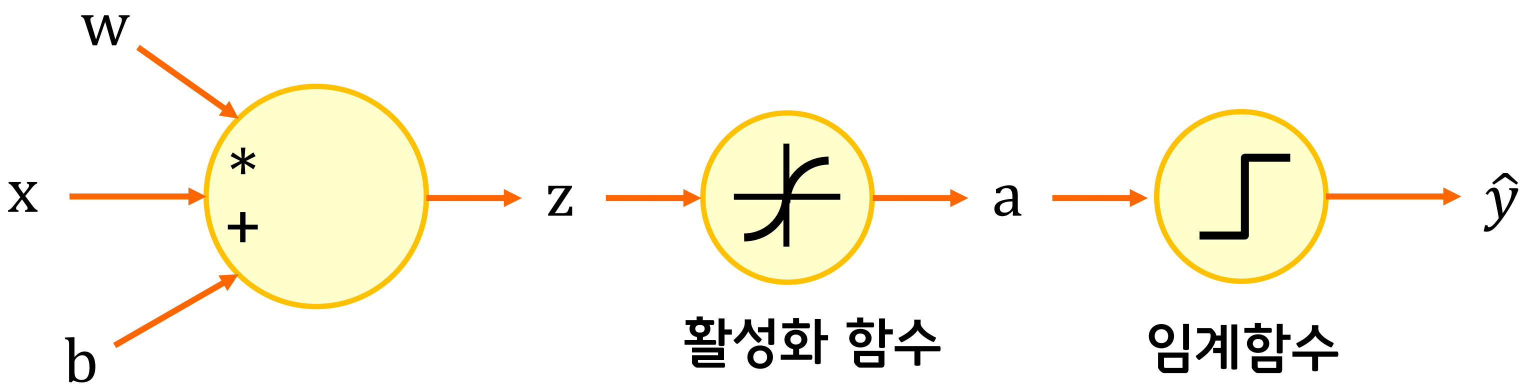
로지스틱 회귀 손실 함수

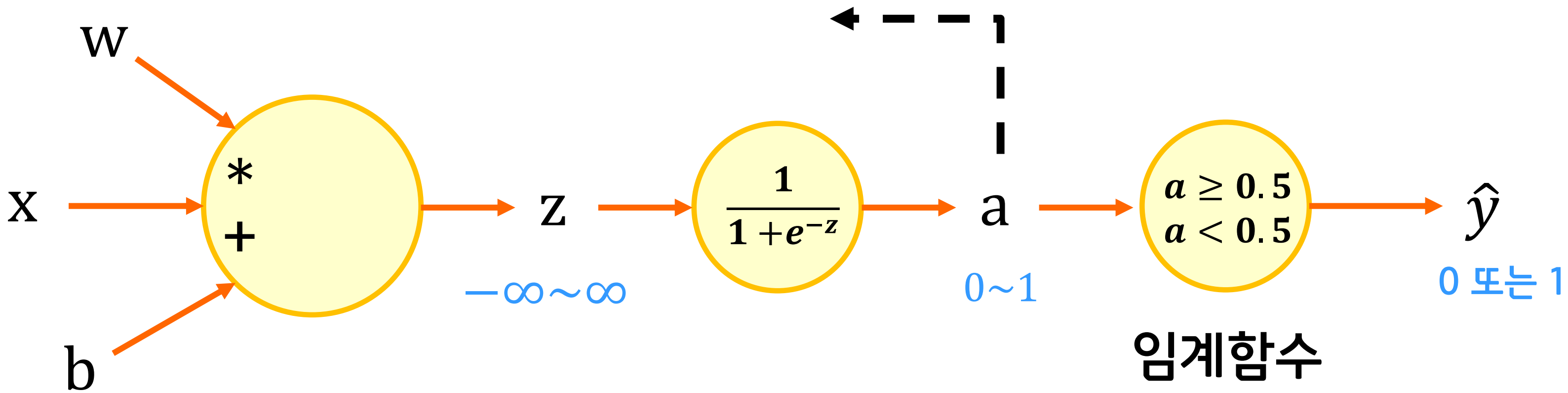
Logistic regression cost function

$$\mathrm{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1 - \hat{y}) & if\ y = 0 \end{cases}$$

$$\mathrm{L} = -(\mathrm{y} * \log(a) + (1 - y)\log(1 - \mathrm{a}))$$

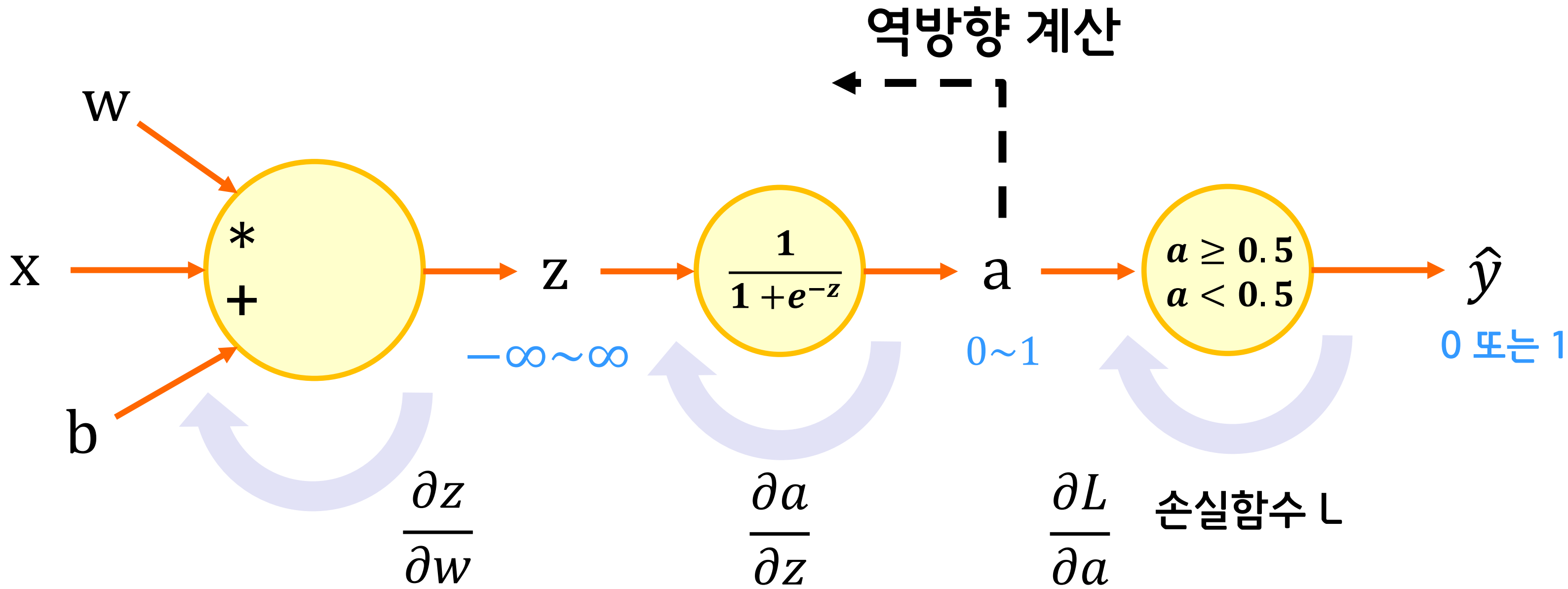
로지스틱 회귀 손실 함수


w
x
b
*
+
z
a
$\hat{y}$
활성화 함수
임계함수


역방향 계산
w
x
b
*
+
z
$-\infty \sim \infty$
$\frac{1}{1+e^{-z}}$
a
0~1
$a \geq 0.5$
$a < 0.5$
$\hat{y}$
0 또는 1
임계함수

특성이 하나인 경우 => x 1개

z = wx + b



chain rule을 **이용하여 각 단계의 미분 결과를 곱한다.**

w에 대하여 미분

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} = (a - y)x$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1 - y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial w} = x$$

b에 대하여 미분

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b} = (a - y)1$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1 - y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial b} = 1$$

특성이 두개인 경우 => x 2개

$y = w_1x_1 + w_2x_2 + b$

$$\frac{\partial}{\partial \hat{y}} \frac{1}{2}(\hat{y} - y)^2 = \hat{y} - y$$

$$\frac{\partial}{\partial w_1} w_1x_1 + w_2x_2 + b = x_1$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1} = x_1(\hat{y} - y)$$

Chain Rule 사용

$$\hat{y} = w_1x_1 + w_2x_2 + b$$

$$J(w_1, w_2, b) = \frac{1}{2}(\hat{y} - y)^2$$

$$\frac{\partial}{\partial w_1} J(w_1, w_2, b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

특성이 두개인 경우 => x 2개

$y = w_1x_1 + w_2x_2 + b$

$$\frac{\partial}{\partial \hat{y}} \frac{1}{2} (\hat{y} - y)^2 = \hat{y} - y$$

$$\frac{\partial}{\partial w_2} w_1x_1 + w_2x_2 + b = x_2$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2} = x_2(\hat{y} - y)$$

Chain Rule 사용

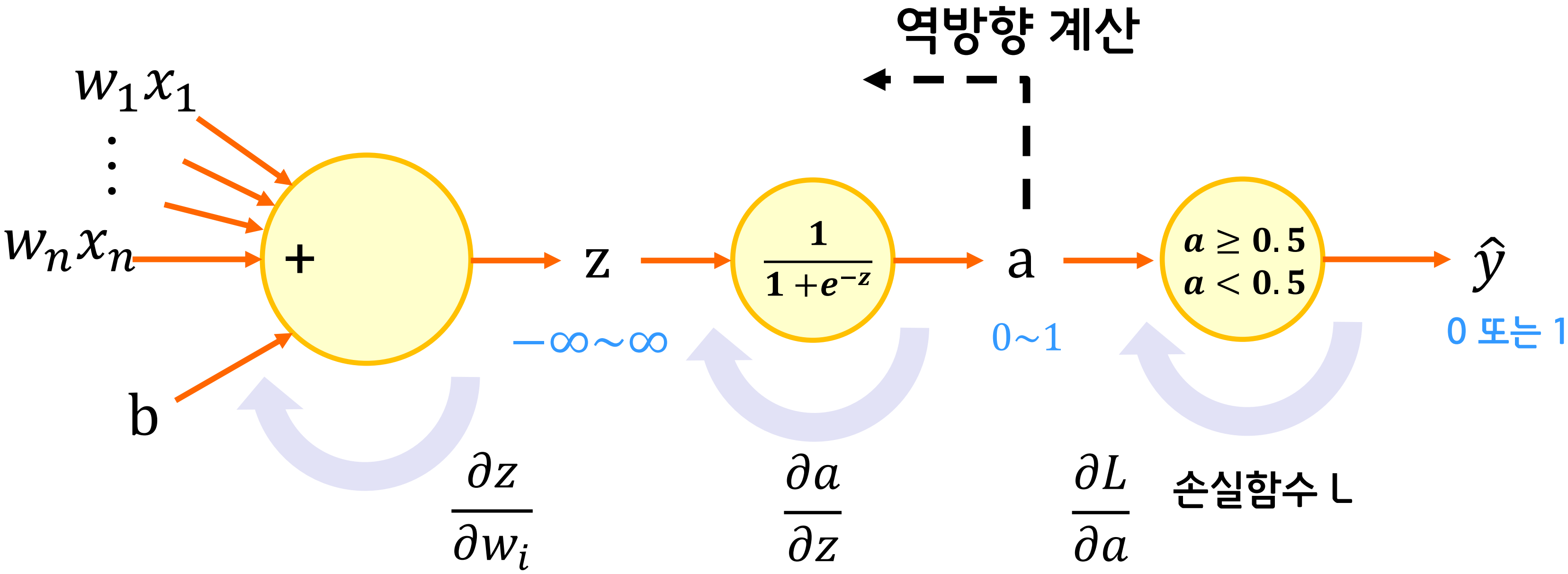
$$\hat{y} = w_1x_1 + w_2x_2 + b$$

$$J(w_1, w_2, b) = \frac{1}{2} (\hat{y} - y)^2$$

$$\frac{\partial}{\partial w_2} J(w_1, w_2, b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

특성이 여러개인 경우 => x n개

$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$



chain rule을 이용하여 각 단계의 미분 결과를 곱한다.

w에 대하여 미분

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i} = (a-y)x_i$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1-y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1-a)$$

$$\frac{\partial z}{\partial w_i} = x_i$$

b에 대하여 미분

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b} = (a-y)1$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1-y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1-a)$$

$$\frac{\partial z}{\partial b} = 1$$

제곱 오차의 미분과 로지스틱 손실 함수의 미분은 $\hat{y}$ 이 a로 바뀌었을 뿐 동일 하다.
따라서 선형함수의 결과 값을 activation 함수인 시그모이드를 적용한 값이 a 이다.

	제곱 오차의 미분	로지스틱 손실 함수의 미분
가중치에 대한 미분	$\frac{\partial SE}{\partial w_i} = (\hat{y} - y)x_i$	$\frac{\partial L}{\partial w_i} = (a - y)x_i$
절편에 대한 미분	$\frac{\partial SE}{\partial b} = (\hat{y} - y)1$	$\frac{\partial L}{\partial b} = (a - y)1$

2. 딥러닝을 위한 수학

2.1 MSE(Mean Squared Error)

2.2 SGD (Stochastic Gradient Descent)

2.3 역전파 구현

2.4 선형회귀 구현

2.5 시그모이드 함수

2.6 로지스틱 회귀 구현

분류용 데이터셋을 준비(위스콘신 유방암 데이터 세트)

	의학	이진 분류
좋음	양성 종양	0
나쁨	악성 종양	1

사이킷런의 datasets 에서 불러온다.

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print(cancer.data.shape, cancer.target.shape)
```

data와 target의 크기를 알아본다.

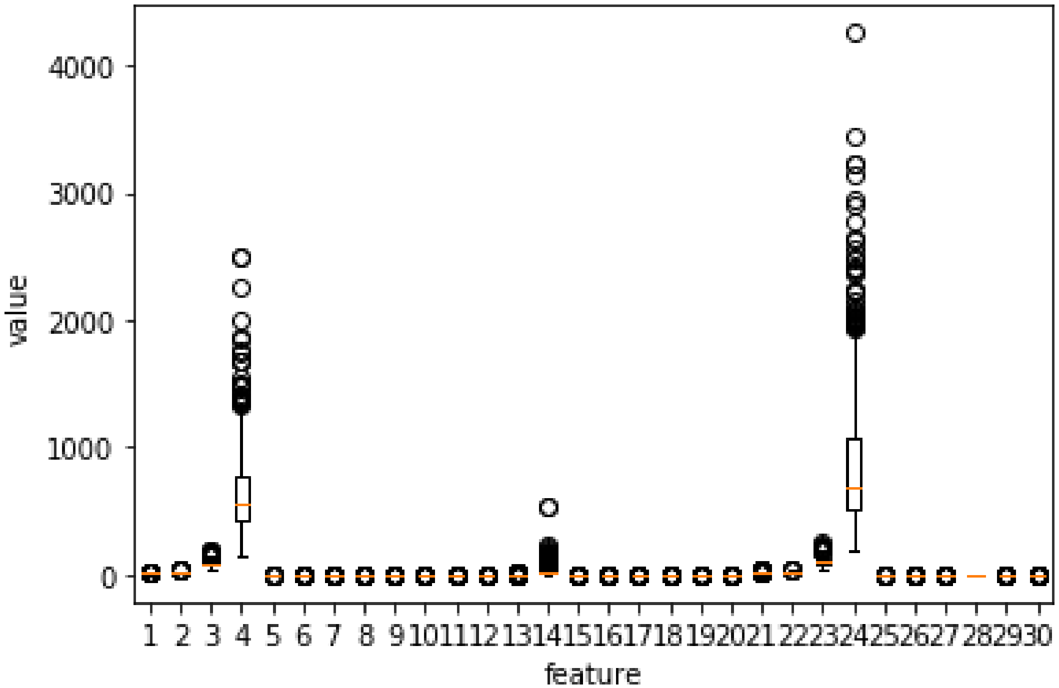
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
print(cancer.data.shape, cancer.target.shape)
```

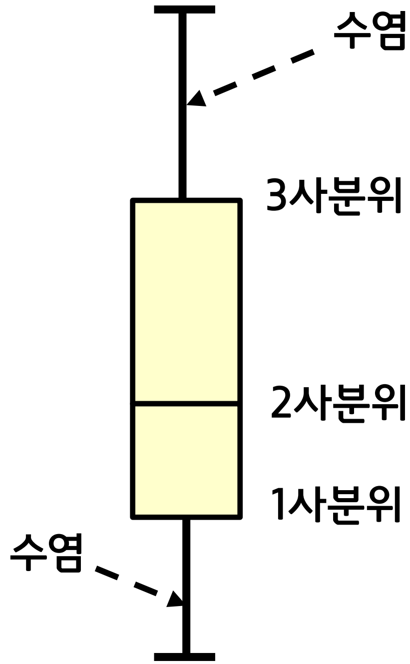
cance에는 569개의 샘플과 30개의 특성이 있다.

```
cancer.data[:3]
array([[1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
        3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
        8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
        3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
        1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, 1.326e+03, 8.474e-02, 7.864e-02,
        8.690e-02, 7.017e-02, 1.812e-01, 5.667e-02, 5.435e-01, 7.339e-01,
        3.398e+00, 7.408e+01, 5.225e-03, 1.308e-02, 1.860e-02, 1.340e-02,
        1.389e-02, 3.532e-03, 2.499e+01, 2.341e+01, 1.588e+02, 1.956e+03,
        1.238e-01, 1.866e-01, 2.416e-01, 1.860e-01, 2.750e-01, 8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, 1.203e+03, 1.096e-01, 1.599e-01,
        1.974e-01, 1.279e-01, 2.069e-01, 5.999e-02, 7.456e-01, 7.869e-01,
        4.585e+00, 9.403e+01, 6.150e-03, 4.006e-02, 3.832e-02, 2.058e-02,
        2.250e-02, 4.571e-03, 2.357e+01, 2.553e+01, 1.525e+02, 1.709e+03,
        1.444e-01, 4.245e-01, 4.504e-01, 2.430e-01, 3.613e-01, 8.758e-02]])
```

박스 플롯을 이용하면 각 특성의 사분위 값을 볼수 있다.

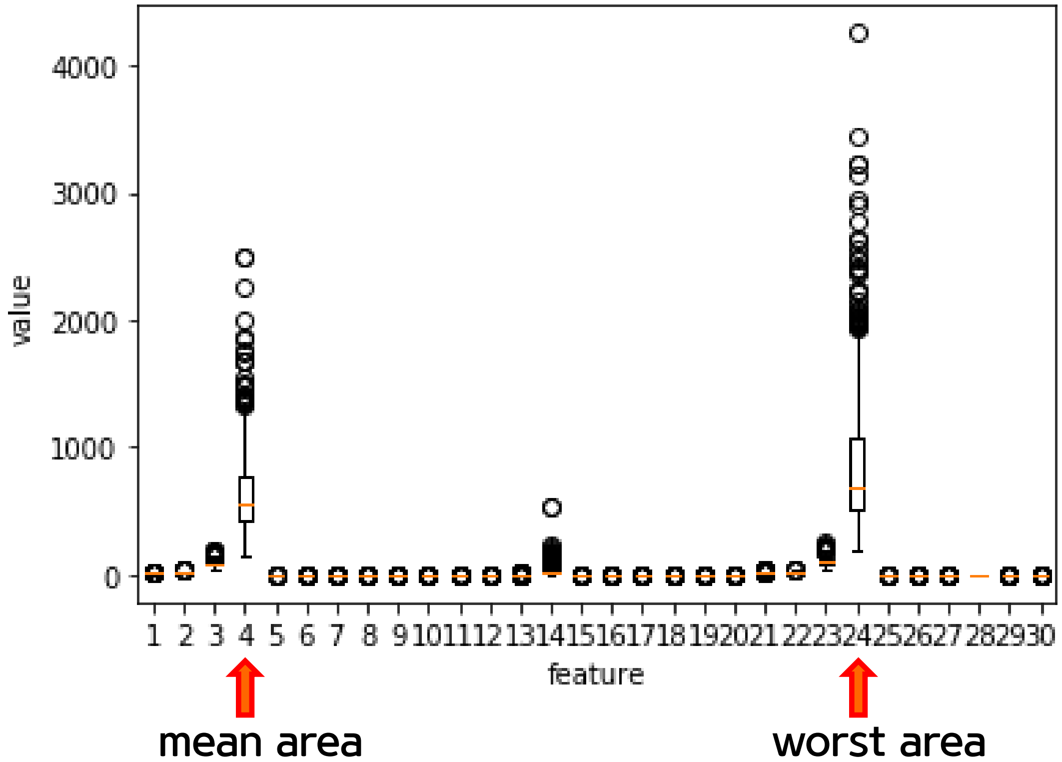
```
plt.boxplot(cancer.data)
plt.xlabel('feature')
plt.ylabel('value')
plt.show()
```





눈에 띄는 특성 보기 : 모두 넓이에 관한 특성이다.

```
cancer.feature_names[[3,23]]

array(['mean area', 'worst area'], dtype='<U23')
```



타겟 데이터 확인

```
np.unique(cancer.target, return_counts=True)

(array([0, 1]), array([212, 357], dtype=int64))
```

넘파이의 unique 함수는 중복 제거후 고유한 값을 찾아 반환한다.

cancer.target에는 0=>악성, 1=>악성 의 값이 있다.

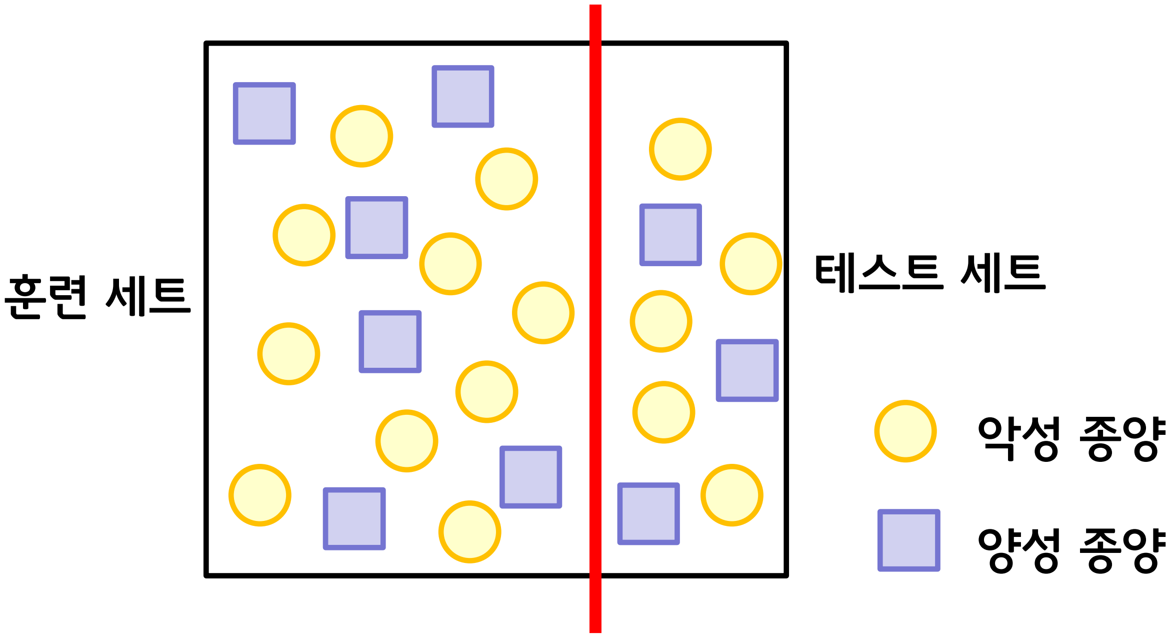
return_counts에는 고유한 값인 0의 개수와 1의 개수가 리턴 되므로

양성종양 212개와 357개의 악성종양 값이 있다.

모델의 성능 평가를 위한 훈련 세트와 테스트 세트

: 모델을 학습 시킨 훈련 데이터 세트로 성능을 평가 하면 상당이 좋은 성능이 나온다.
하지만 이렇게 하면 '과도하게 낙관적으로 일반화 성능을 추정한다' 라고 한다.
따라서 전체 데이터를 나누어 훈련 데이터 세트로 학습시킨 후 테스트 데이터로 성능을 평가 해야 한다.

주의 사항
- 보통 훈련 데이터가 더 많아야 하므로 8:2 정도로 나누어 훈련과 테스트를 진행한다.
- 데이터가 어느 한쪽에 몰리지 않도록 비율이 일정하게 골고루 섞어야 한다.



타겟 데이터 확인

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y,
test_size=0.2, random_state=42)
```

훈련 데이터와 테스트 데이터의 비율을 8:2로 설정한다.

난수 초깃값 42로 지정하여 무작위로 데이터 세트를 섞은 다음 나눈다.

훈련 데이터를 나눌때 class의 비율을 동일하게 만든다.

8:2 나눈 비율 확인

```
print(x_train.shape, x_test.shape)
(455, 30) (114, 30)
```

data 비율 유지 확인

```
np.unique(y_train, return_counts=True)
(array([0, 1]), array([170, 285], dtype=int64))
```

로지스틱 회귀 구현

이전 코드에서 바뀐 코드 설명

```
def __init__(self):
        self.w = None
        self.b = None
```

특성이 하나가 아니라 여러개가 될수 있으므로 __init__함수에서 초기화 하지 않고
실행중 x에 입력되는 특성의 개수에 따라 초기화 한다.

```
np.sum(x * self.w)
```

forpass함수의 해당 코드는 x가 더이상 하나의 값이 아니라 30개의 특성을 가지고 있는 넘파이 배열이므로
np.sum(x * self.w) => x[0]*w[0] + x[1]*w[1] + ... + x[29]*w[29] 로 해석 된다.

```
self.w = np.ones(x.shape[1])
```

fit 함수의 w 초기화는 x.shape[1]의 값이 30 이므로 넘파이 배열의 개수를 30개로 할당 하고
ones 함수에 의해 각 원소는 1로 초기화 된다.

```
def activation(self, z):
        a = 1 / (1 + np.exp(-z))  # 시그모이드 계산
        return a
```

activation 함수를 구현하고 시그모이드 함수의 결과 값을 리턴한다.

이전 코드에서 바뀐 코드 설명

```
def predict(self, x):
        z = [self.forpass(x_i) for x_i in x]    # 정방향 계산
        a = self.activation(np.array(z))        # 활성화 함수 적용
        return a >= 0.5
```

예측 함수를 구현한다.

z = [self.forpass(x_i) for x_i in x] 코드는 파이썬에서 지원하는 코드로 대괄호([]) 안에 for in과 함수의 호출부를 넣을수 있다. 이때 동작은 x의 각 원소를 x_i에 뽑아서 x_i를 forpass의 인자로 전달하여 forpass의 리턴값을 list로 append 하여 x의 가공된 값을 z에 대입한다.

np.array(z) 함수는 list를 넘파이 배열로 바꿔준다.

a에는 최종적으로 시그모이드 함수의 결과가 대입되므로

결과값이 0.5 보다 크다면 악성종양으로 판단하여 1을 리턴한다.

```
np.mean(neuron.predict(x_test) == y_test)
0.8245614035087719
```

np.mean 함수는 평균을 계산하는 함수로 예측값과 실제값의 비교 값을 평균을 계산하여 정확도를 측정한다.